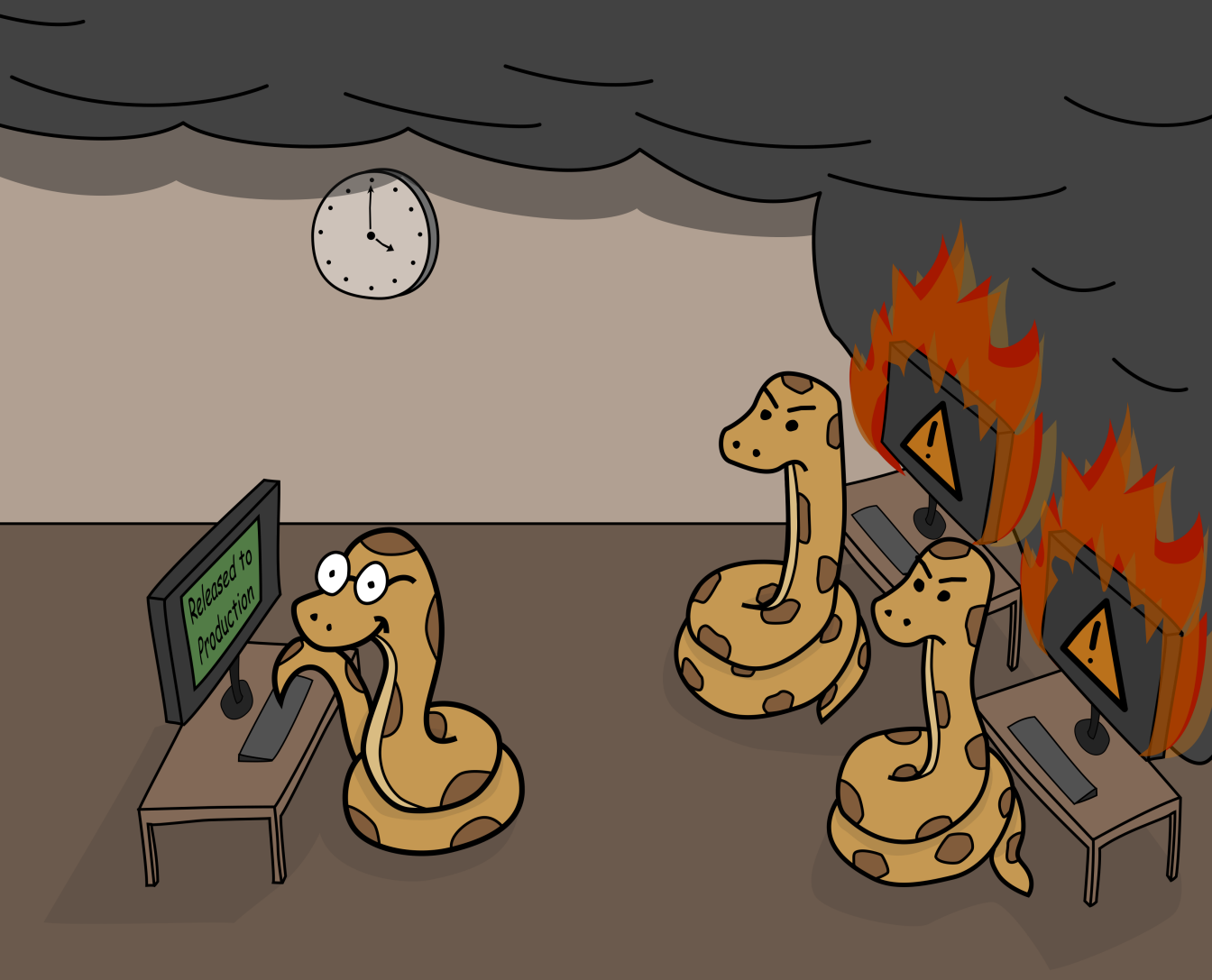
PRACTICAL PYTHON

FOR

PRODUCTION UNDER PRESSURE



Alex Telford

2025

Practical Python for Production under Pressure

Managing the insanity that is developing pipelines alongside a production at full speed

Alex Telford

This book is available at https://leanpub.com/practical_python

This version was published on 2025-05-13

This book is dedicated to my lovely wife and children who have tolerated many years of my production induced insanity.

Contents

Practical Python for Production under Pressure

> Production tools must exist and work at the time they are needed. Anything that compromises this will inevitably make the tool unsuitable for production.

Introduction

Developing tools and pipelines for use in Film and Games is a unique challenge, where we must finely balance speed, stability, usability, and performance. Everything is always urgent, everything is always broken, and we developers are left scrambling to hold the system together while begging production for time to "do things properly" and clean up accumulated code debt. In a sense, it's like being in a pit crew; getting the driver back out there is your number one priority, sometimes with more duct tape than we care to admit.

That is what this book is about: how to deliver better tools while facing the uphill battle of a production in full swing and hopefully retaining some semblance of sanity at the end.

This book is aimed at TDs with some experience with Python and Qt/PySide and an interest in pipeline, but it will be mostly theory and workflows rather than a deep dive into code. It started as an article focused on Python in production but grew into a more theoretical viewpoint, this means it's driven larger by my personal opinions as a self-taught engineer. It is also important to note that this book focuses on techniques we use in the fast-paced production environment and often does not correlate to "Python best practices."

Why Python?

This question comes up more often than I'd like: Python is slow, it's dynamically typed, it's not secure...

So why do we use it?
Because it's the right choice for the situation.
Python is like the play-dough of scripted languages; sure it's a bit unstructured, but that means it can morph and change as needed. And oh boy do productions like to change things under us.

Python is chosen not because of its performance, but its flexibility, integrations with existing platforms, its low barrier to entry, and ease of distribution.
Also, historical inertia.
Way back when, these modern approaches did not exist, and low-level languages like C/C++/Java/etc. are not exactly tech-artist friendly. After a series of DCC-specific languages, eventually Python became the standard, now so much has been built upon it it's become the standard language for VFX pipelines.

This does not mean that Python is the "best" language, and of course you can build pipelines in other languages, but as of now, if you want to write pipelines for production then Python is a great place to start.

About the author

Kia ora koutou,
I am Alex Telford, a Senior Pipeline and Software Engineer from Paraparaumu, New Zealand. I have been working in the VFX and Games production world for a little over 12 years; prior to that, I worked as an artist doing graphic design, websites, and short commercials. I have experience both working side by side with artists on the frontline of productions in panic mode and developing long-term software in larger engineering teams.

I don't know everything, and what I do know is up for debate

I've been doing this a while, and while I've picked up many skills and techniques over the years, I am regularly corrected on my misconceptions; by juniors and interns just as much as seniors. I am a self-taught engineer with an eclectic and random collection of knowledge. In fact, even while writing this book I was forced to confront some of my own misunderstandings.

When reading this book, I don't want you to blindly follow my words; I want you to internalize them and challenge them. We all come from different backgrounds and experiences, and those varied perspectives are what help

drive ourselves forward. If you find you disagree with a point, that is fantastic. Call me out and let's discuss it.

A note on LeanPub

This is my first time using LeanPub (and writing a book) and naturally there are going to be a few quirks. One issue is code blocks wrap by adding a backslash "\" at 80 characters or so. While I've tried to work around this if you find you get a syntax error when copying a code snippet, check there aren't any errant backslashes first.
Leanpub also indents the first line of every paragraph on the free tier, I've cleaned this up but can upgrade to pro if there is interest.

Getting the resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

"Frontline" tools and Code Debt

Personally, I have built hundreds of tools over the years and one thing I can say for sure is that the closer you are to the "frontline" (Working directly with the artists), the more insane and panicked things get with an emphasis on speed over stability. While on the opposite side of the spectrum, developing tools for external customers, stability takes precedence with pressure on getting things right over just making it work.

This rapid development creates something we call "Code Debt"; In order to meet deadlines and get production moving we create problems for developers and artists further down the line, particularly when our tools outlive their intended period of use.
For example these problems may occur:

- Code may catastrophically fail because a dependency or rig setup changed, resulting in crashes and lost work.
- Code turns into a Frankenstein of hacked together updates that becomes unmaintainable even by it's original developers.

- Code continues to work but is so slow it costs artists hours of time
- Code still works, but no one knows exactly what it does and it gets encased in wrappers embedding itself into unstable permanence.

* * *

A long time ago, I was working with some code artists used to merge scenes together, it was slow, sometimes it took hours, sometimes even days, but we used it because it worked and it worked well.

One day I was asked to speed it up and dive into this franken-monster of a code base to figure out what it was doing. After a lot of searching we found that it was spending 90% of its time running checks and gathering information that was discarded without ever being used, this had cost the company thousands of wasted artist hours before being detected and because

the code was undocumented we still don't know what that data was originally for or why it stopped being used.

Another memorable example of code debt is when I wrote a tool to export cameras, sent the prototype to test and forgot about it because I never heard back and it got lost amongst the dozens of other odd tasks I had going.
Well 5 years later a different department contacted me because it broke, this tool had somehow been passed around by artists and kept working for years as a way to work around the official publishing tools.
This meant we potentially had years of bad data being pushed around, fortunately this was easily resolved but could have been much worse.

This goes to show that code debt raises its ugly head in more ways than just messy code; it can have real impacts on artist time and data integrity.

So why do we continue to develop this way?

In short, poor planning, both on our part and on production's. We aren't asking the right questions at the start, aren't tracking things correctly, and we aren't given enough time to develop every tool to a high standard.

However, that is no excuse to release crappy code. We have a responsibility to do what we can in the time we have to deliver code that produces minimal debt, and if you are feeling pressured to deliver code you know will cause problems down the line, you need to speak up.

Yes, we will still produce code debt, but by planning ahead and utilizing our time well, we can drag that project over the finish line without leaving a huge mess to clean up later.

The project comes first

This is a tough one for developers used to working in traditional development teams. The project has to be delivered; we often simply don't have time to do it properly. It sucks, we are essentially forced to lower our standards and create tools that are barely functional in order to keep things moving.

But this is ok, remember that we aren't delivering software, we are delivering productions. There are going to be times that artists need both urgent and long time solutions.

* * *

As an example, a while back I had to build a tool to handle arbitrary submissions to Shotgrid for an urgent client request. This was something we were going to need to use for the next few months with this client but they also needed it right now to get the first client deliveries out.

We had the shotgrid and ffmpeg apis ready to go, but with the many options we knew we needed a UI.
I split the task into two stages:

- First was an roughed together UI that did the job, albeit barely. This would tide the team over for the next week or two
- Next I made a priority task to create a properly designed and tested UI that would operate over the next few months.

Naturally once the rough UI was in place we're of course asked to drop the feature-complete version in favour of other critical projects, but nope, not having it. They agreed to the two stage process and that is what's happening now.

This is a fairly standard situation, it's easy to just cobble together a solution and then blame production for instabilities, or deny a request entirely because you cannot deliver a sustainable result in a short timeframe. But we have to take the big picture into account. Yes it's more work to build the same tool twice, but we have to work with what we have.

Communication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Preparing for a task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Clarifications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Planning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

My Approach

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

My first question is:

Next, Do I have the required code to do this?

The unspoken requirements:

Implementation:

Task 2

My Approach

Questions for the users:

Questions for the department lead:

The unspoken requirements:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Implementation:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

API Requirements:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

General workflow:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

User Interfaces:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Further tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

My Approach

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Questions for the artists:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Questions for the department:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Profiling:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Identifying key areas to address

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 4

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

My Approach

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Questions for the artists:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Questions for the dev team

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

My Approach

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Questions for my Project Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Questions for the stakeholders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Managing your workload

Production environments are high pressure; there is far more work than we can ever hope to accomplish, and no matter how much extra work we do or last-minute requests we fulfill, it's never enough.
There is a reason this industry is prone to burnout and misery. Yes, a part of it is the unnecessary pressure we receive, but ultimately we are just as complicit in this.

We are just too damn helpful; most TDs and developers I've met working closely with production seem to have the same unhealthy desire to help people. Sure, we complain and blame others for being idiots or not understanding what we do and making our life hard, but deep down we want to be helpful.
This is where communicating and tracking are indispensable tools.

Communicating

The first step in managing your workload is to actually talk about it, others don't know what your limit is unless you voice it. It's in the companies best interest to find the balance between productivity and stress, which is different for each person.

Know your capacity

Ideally, you should be operating at around 50-70% capacity on any given day regardless of how busy production is.
There is no such thing as "giving 110%". Even giving 100% would mean giving up food, coffee and social connections.

Rest is important and needs to be factored into your day, especially for developers like ourselves with high mental loads.
If it helps you can think of "Giving 100%" as including breaks, this is critical thinking time.

> "We are paid to think really hard about things. Sometimes we write it down."

You also need to allocate time to be able to pace yourself; design, iteration and testing are important parts of the process but often the first to go when

we are pushed to our limits.
This is why we need a respectable buffer in our day, it doesn't mean we aren't working hard, we are just working in a sustainable fashion.

A good rule of thumb is to allocated half your day as "work", the other half will fill with meetings, breaks, discussions and other shenanigans that always crop up.
Also, most of us are horrible at estimating how long something actually takes, so it's always better to under promise and over deliver.

No is always a valid answer

There is something you need to know.
"No" is always a valid answer.

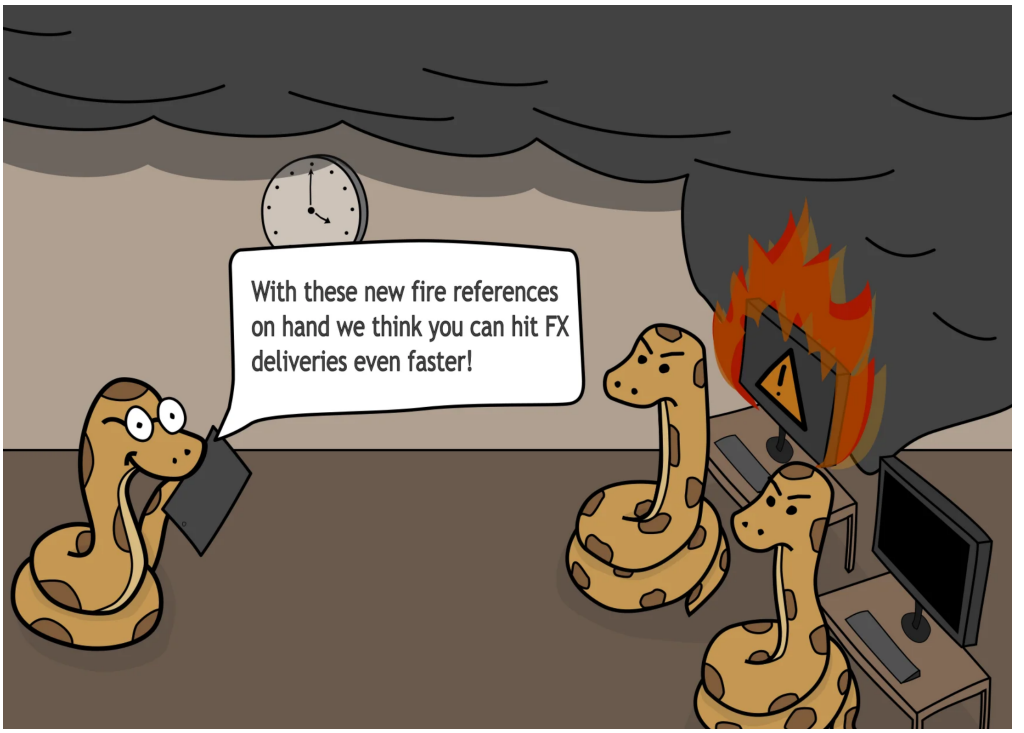


Figure 1. No

Getting pressured into creating something you know will cause problems or to take on more work than you feel comfortable with is far too common in this industry.

Similarly, every time you get pulled out of the headspace of your task can add up to 20 minutes of disruption to get back into it, you can say no to even discussing something if you are busy, whatever it is it can probably wait.
It doesn't have to explicitly be "No" either, there are many ways you can voice your concerns about a project or workload or simply defer it until later, for example:

- Hey I'm really busy with these other tasks, but lets put it on the backlog and look at slotting it in a future sprint.
- I'm not sure that's going to fit in with X, have you considered using Y instead for this?
- That is potentially a lot of work to do properly and will take weeks of work to design, build test and document. Can you check in with (stakeholder/manager) that we want to devote this much time on this?
- We really need to get some tests on this before we deploy, I'm not convinced it's stable yet and we don't want to break artists scenes.
- Sorry I've really got to stay in the headspace of this task, I'm free around 2pm, let's discuss it then.

Know your limits and if you are feeling the pressure, talk to someone about it. A good lead is someone who protects their team from undue outside influence. If not a lead, then find a Senior or mentor to talk to.

Getting Feedback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Actually Getting Feedback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So how do you get feedback?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

There are a few things you can do

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What should you not do:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Impostor Syndrome

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Managing Environments with Rez

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Why rez?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Setting up Rez

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What is rez

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Allzpark / Bleeding-rez

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Installing Rez

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Creating a Config

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Rez on Windows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Installing external packages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Compiling external packages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Fixing PySide

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Adding additional python versions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Resolving an environment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Creating custom packages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

package.py

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Requires

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Commands

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

build.py

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Releasing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Production, Staging and Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Ephemerals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting DCC included dependencies to play nice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structuring your project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Private until it's needed.

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So what belongs in the public API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Separating your code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Creating maintainable APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

One class per file

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use specific file names

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use typing where possible

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Abstract Interfaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

When to not use interfaces:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Data Structs

A common issue with code that grows over time is endless keyword arguments, does this look familiar to you?

```
def publish_data(self, start:int, end:int, notes:str, items,
                 client_notes:str=None, user:str=None,
                 verbose:bool=False, mode=None):
    some_code_here
```

Perhaps the method started with a frame range, notes and items to publish, later we were asked to add client notes, then some other options. Now the order is messed up and no one knows what mode does.
What's worse is when these parameters need to be passed along the chain to other methods.
I have seen pipelines that pass 20+ arguments through 10+ functions, every time a new option was needed it had to be added to all of them and tested accordingly.

This can be made even worse when a well meaning developer changes the arguments to **kwargs, this gives freedom to pass any data without updating every function in the chain, but you lose all track of what data is actually used and where it goes.
This is where structs, aka: dataclasses, come in handy.
A dataclass is as it sounds, a collection of related data that can be passed around in an intuitive manor.
So the above method might change to this:

```
from dataclasses import dataclass, field
from typing import List
import abc
import enum
import os

class PublishMode(enum.IntEnum):
    Default = enum.auto()
    DryRun = enum.auto()
    ExtendedBakes = enum.auto()

class IPublishableItem(abc.ABC):
    @abc.abstractmethod
    def node(self)->str:pass

    @abc.abstractmethod
    def type(self)->str:pass

    @abc.abstractmethod
    def entity(self)->str:pass

```

```
@dataclass(frozen=True)
class FrameRange:
    start:int
    end:int
    step:float = 0.0

@dataclass(frozen=True)
class PublishOptions:
    frame_range:FrameRange
    notes:str
    client_notes:str=""
    user:str=field(default_factory=os.getlogin)
    verbose:bool=False
    publish_mode:PublishMode=PublishMode.Default

def publish_data(self, items:List[IPublishableItem],
                 options:PublishOptions):
    some_code_here
```

Here the code is explicit in what options are required and what is optional, we only have two types of data being passed in and adding additional options later without having to edit the entire pipeline is a breeze.

We can also "freeze" a dataclass to ensure that it cannot be modified after it is created which helps with ensuring data consistency, this is super helpful when preventing functions in a pipeline modifying data out of turn.
Just like with interfaces though, they are not correct for every situation as they do add a small amount of overhead to calling code; particularly when you have a structure that only has one use you might want to think first about breaking it into multiple datastructures or just using arguments.

Excercise

Here is an example function with a bunch of parameters, because this function has been modified throughout the years the parameters are not in order and have special relation to each other.
Your task is to refactor this to have clear and organized parameters for readability and ease of use.
Because this function exists within a larger pipeline, you must keep the original function and call your new one with the parameters.

```python
from pathlib import Path
import os

def export_client_package_data(
    source_scene:Path,
    project:str=None,
    tree:str=None,
    scene:str=None,
    shot:str=None,
    asset:str=None,
    variant:str=None,
    shot_start:int=None,
    shot_end:int=None,
    username:str=os.getlogin(),
    frame_handles:int=0,
    export_format:str="csv",
    entities=None,
    notes:str=None,
    preroll:int=0,
    target_directory:Path=None,
    dryrun:bool=False,
    test:bool=False,
    verbose:bool=False,
)->Path:
    """ Exports a spreadsheet of data for the client package

    Args:
        source_scene: path to the maya scene file to export
        project,tree,scene,shot,asset,variant: The context to export to
            If not specified, this will be gathered from where the
            source_scene path is. either scene/shot or asset/variant
            can be specified else this throws an error
        shot_start/shot_end: the frame range for the shot
        frame_handles: extra frames to add at the start and end
        preroll: extra frames added at the start for bake info
        username: the artist who created this package
        export_format: spreadsheet format, supports csv or xlsx
        notes: client notes, NOT artist notes
        target_directory: where to export this spreadsheet to,
            errors if not specified
        entities: list of entity data in the format:
            [{name:str, asset:str, variant:str, type:str, number:int}]
        dryrun: doesn't create the file, just prints the data,
            this allows target_directory to be None
        test: if True will set target_directory to a tempdir
        verbose: if True will print the data before saving

    Returns:
        path to the spreadsheet file
    """
```

```
    pass
```

Solution

```
from pathlib import Path
import os
from dataclasses import dataclass
from typing import List, Dict, Optional


@dataclass(frozen=True)
class Context:
    project: Optional[str] = None
    tree: Optional[str] = None
    scene: Optional[str] = None
    shot: Optional[str] = None
    asset: Optional[str] = None
    variant: Optional[str] = None


@dataclass(frozen=True)
class FrameRange:
    start: Optional[int] = None
    end: Optional[int] = None
    frame_handles: int = 0
    preroll: int = 0


@dataclass(frozen=True)
class EntityData:
    name: str
    asset: str
    variant: str
    type: str
    number: int


@dataclass(frozen=True)
class ExportClientPackageParams:
    source_scene: Path
    context: Context = None
    frame_range: FrameRange = None
    target_directory: Optional[Path] = None
    entities: Optional[List[Dict]] = None
    notes: Optional[str] = None
    username: str = os.getlogin()
    export_format: str = "csv"
    dryrun: bool = False
```

```
    test: bool = False
    verbose: bool = False

    def __post_init__(self):
        # You can add validations here, like checking context
        # and range is set correctly.
        pass


def export_client_package_data_v2(params: ExportClientPackageParams)->Path:
    """ Exports a spreadsheet of data for the client package

    Args:
        params: ExportClientPackageParams

    Returns:
        path to the spreadsheet file
    """
    pass


def export_client_package_data(
    source_scene: Path,
    project: str = None,
    tree: str = None,
    scene: str = None,
    shot: str = None,
    asset: str = None,
    variant: str = None,
    shot_start: int = None,
    shot_end: int = None,
    username: str = os.getlogin(),
    frame_handles: int = 0,
    export_format: str = "csv",
    entities = None,
    notes: str = None,
    preroll: int = 0,
    target_directory: Path = None,
    dryrun: bool = False,
    test: bool = False,
    verbose: bool = False,
) -> Path:
    context = Context(
        project=project,
        tree=tree,
        scene=scene,
        shot=shot,
        asset=asset,
        variant=variant
    )
```

```
95
96      frame_range = FrameRange(
97          start=shot_start,
98          end=shot_end,
99          frame_handles=frame_handles,
100         preroll=preroll
101     )
102
103     params = ExportClientPackageParams(
104         source_scene=source_scene,
105         context=context,
106         frame_range=frame_range,
107         target_directory=target_directory,
108         entities=entities,
109         notes=notes,
110         username=username,
111         export_format=export_format,
112         dryrun=dryrun,
113         test=test,
114         verbose=verbose
115     )
116
117     # Call the refactored function
118     return export_client_package_data_v2(params)
```

Decorators

Exercises

Task 1

Solution

Task 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Other Tips

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Stacks and Recursion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Cross-Platform development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use pathlib for paths

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Schema

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

When should you use Schema?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Json-Schema

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Extra Properties

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

array

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

number

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

integer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

string

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

boolean

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

enum

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Defaults

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Defs and Refs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

JSON5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Versioned Schema

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Designing with Diagrams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structure Diagram

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Flow Diagram

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Creating maintainable Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

The flexibility tradeoff

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

The data handshake

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Nodes and Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Mapping it out

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Refactoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Before you start

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Questions you need to answer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Examples:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task: Add Houdini support to the Asset Publisher UI.

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task: Add usd support to the camera publisher.

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Finding usage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Regression Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

After you finish

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Maintaining concurrent APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Common Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Built-ins

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

pathlib

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

dataclasses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

itertools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

collections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

functools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

attrs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

box

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

PyDantic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Algorithms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

numpy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

pandas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

matplotlib

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

jsonschema

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

pyYAML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

pyjson5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Automated tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Types of tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Unit/Integration tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Regression tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Interaction tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

End-to-end tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Writing Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Common best practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

A simple unittest example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

A simple pytest

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

conftest.py

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

pytest plugins

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Test Driven Development (TDD)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So should you use it?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Guided Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Test camera publish:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Production Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Production Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Schema is important

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Load Balancing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Paging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Track your sources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Secure your api keys

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Flow/Shotgrid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Caveats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Entities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Schema

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Optimizing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Pre-Cache your data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use combination filters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Comparison Operators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use Threading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

FTrack

FTrack is a relational database-based tracking tool similar to shotgrid but uses a more "sql" style syntax for querying data.
The best way I've heard FTrack compared to Flow is like Mac vs Linux. FTrack works out of the box but doesn't have the depth of configuration that Flow has. That said, what FTrack has, is excellent developer documentation.

API

FTrack has three APIs you can use, the Python API:
https://developer.ftrack.com/api-clients/python/
The Javascript API:
https://developer.ftrack.com/api-clients/javascript/
And of course the api endpoints directly:
https://developer.ftrack.com/api/endpoint

Mostly we will use the python API unless you have a specific reason to use another api (like using ftrack from C++ or Javascript)

Caveats

- As with all these production APIs, the FTrack python API is not threadsafe, you will need appropriate locks around queries/commits or use a new session for each thread.
- FTrack has a caching system built in, this can be extended for more persisteny caching but may result in out of date data at times.
- FTrack optimizes to not query data as it's needed, this results in network queries and latency where you may not expect it.

Entities

In FTrack, an entity is a dict-like object that represents the current state of the data.
Where this differs from Flow is that these may include local changes not yet sent to the server.
What's neat is that the FTrack API dynamically creates class structures on connection to the server based on your current schema.
This allows you to create custom entity types in FTrack without additional wrapper overhead.

One thing to be aware of is that these entities are "live", you can query updated data and update the data from the entity itself which can be committed later with session.commit().
While this does make batch updating convenient it does also mean that data may go out of sync if there is a long time between editing and committing.

Schema

Like Flow, FTrack is heavily schema based and this can be queried via the API.
While not explicitly mentioned, I believe FTrack uses json-schema specification based on the structure.
For example, here is the schema for Note:

```
{"default_projections": ["id"],
 "id": "Note",
 "immutable": ["id"],
 "metadata": {"entity_event": True, "group_by": True},
 "primary_key": ["id"],
 "properties": {"author": {"$ref": "BaseUser", "alias_for": "user"},
                "category": {"$ref": "NoteCategory"},
                "category_id": {"alias_for": "categoryid",
                                "description": "Foreign key(s): "
                                               "NoteCategory.id",
                                "format": "foreign_key",
                                "type": "string"},
                "completed_at": {"format": "date-time", "type": "string"},
                "completed_by": {"$ref": "User"},
                "completed_by_id": {"description": "Foreign key(s): "
                                                   "User.user_id",
                                    "format": "foreign_key",
                                    "type": "string"},
```

```
                "content": {"alias_for": "text", "type": "string"},
                "date": {"format": "date-time", "type": "string"},
                "frame_number": {"type": "integer"},
                "id": {"alias_for": "noteid",
                       "default": "{uid}",
                       "type": "string"},
                "in_reply_to": {"$ref": "Note", "alias_for": "noteparent"},
                "in_reply_to_id": {"alias_for": "noteparentid",
                                   "description": "Foreign key(s): Note.id",
                                   "format": "foreign_key",
                                   "type": "string"},
                "is_todo": {"default": False, "type": "boolean"},
                "metadata": {"alias_for": "meta",
                             "items": {"$ref": "Metadata"},
                             "key_value_attributes": {"key": "key",
                                                      "value": "value"},
                             "type": "mapped_array"},
                "note_components": {"items": {"$ref": "NoteComponent"},
                                    "type": "array"},
                "note_label_links": {"items": {"$ref": "NoteLabelLink"},
                                     "type": "array"},
                "parent_id": {"type": "string"},
                "parent_type": {"transform": {"client": "{schema_id}",
                                              "server": "{table_name}"},
                                "type": "string"},
                "project": {"$ref": "Project"},
                "project_id": {"description": "Foreign key(s): "
                                              "Project.project_id",
                               "format": "foreign_key",
                               "type": "string"},
                "recipients": {"alias_for": "note_recipients",
                               "items": {"$ref": "Recipient"},
                               "type": "array"},
                "replies": {"alias_for": "notes",
                            "items": {"$ref": "Note"},
                            "type": "array"},
                "thread_activity": {"alias_for": "triggerdate",
                                    "format": "date-time",
                                    "type": "string"},
                "user_id": {"alias_for": "userid",
                            "description": "Foreign key(s): BaseUser.id",
                            "format": "foreign_key",
                            "type": "string"}}},
    "required": ["id", "user_id"],
    "system_projections": ["project_id", "parent_id", "userid"],
    "type": "object"}
```

There is a lot of information here, This data is not really made for us, it's used

internally by FTrack.
Below is the code that pulls this data from FTrack:

```
## Get a list of entity types
entity_types = sorted(session.types.keys())
for entity_type in entity_types:
    print(f"- {entity_type}")

## Find the details for a specific schema
for schema in session.schemas:
    if schema.get("id") == "Note":
        pprint.pprint(schema)
        break
```

If you need to access the schema of a specific entity, just grab it of the dynamically generated object:

```
session = ftrack_api.Session(
    api_user=USER,
    api_key=API_KEY,
    server_url=DOMAIN
)
project = session.types["Project"]
attr = project.attributes.get("full_name")
print(attr.data_type)
print(attr.default_value)
```

Optimization

While FTrack cleverly hides this behind the API, ultimately it is direct database queries.
This means that you need to explicitly state which fields you are wanting to query otherwise it will query them on demand as needed.
Unless you are bashing quick scripts together, I recommend you turn this feature off.

```
session.auto_populate = False
```

Ideally, your code should be very explicit about what fields it needs without

just leaving it to the server to query them as needed, else you may introduce additional latency to your tools. Particularly if your server is under significant load.
This means you need to add the fields in your query:

```
project = session.query(
    "select full_name, start_date from Project where name is 'sync'").first()
print(f"Project: {project['full_name']}, {project['start_date']} "
      f"(ID: {project['id']})")
```

Any field you don't query other than 'name' and 'id' will be set to "NOT_SET". If you want all the fields, you will need to explicitly pass them as FTrack does not support * for all fields:

```
project_fields = list(session.types["Project"].attributes.keys())
fields_str = ', '.join(project_fields)
project = session.query(
    f"select {fields_str} from Project where name is 'sync'").first()
```

Though outside of debugging I wouldn't recommend this as you are explicitly requesting far more data than necessary, the local processing may result in more time than you save in latency.

Combination Filters

We don't really get the combination issue in FTrack like we did in Flow. This is because the query language is so explicit and documented it would be insane to write code like this:

```
project = session.query("Project where name is 'sync'").first()
sequence = session.query("Sequence where name is 'BikeChase'"
                         f" and project_id is {project['id']}").first()
shot = session.query("Shot where name is 'bc0060'"
                     f" and parent_id is {sequence['id']}").first()
```

Instead of:

```
shot = session.query("Shot where name is 'bc0060'"
                     " and parent.name is 'BikeChase'"
                     " and project.name is 'sync'").first()
print(shot)
```

That said, there are some performance tweaks you can use.
First is combining relation queries together, if you need to query multiple attributes from a linked field then instead of specifying all individually, group them together with “relation has (conditions)”

```
shot = session.query("Shot where name is 'bc0060'"
                     " and parent has (name is 'BikeChase' and bid < 1)"
                     " and project.name is 'sync'").first()
```

This just optimizes the server side query.

Kitsu

Local Installation

API

Caveats

Entities and Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Queries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Pre-Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use Threading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Codecks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Python API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Shenanigans

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting an auth token

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Jira

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Searching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Issues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Updating

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Comments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Wrapping APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

When not to wrap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Wrapping an API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Backend

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Adapters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Client

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

PDB

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Common pdb Commands

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Printing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Executing code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Stepping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Extending PDB

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Traceback_with_variables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Inspecting the exception stack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Hooks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Inspecting

Tracking Objects

Often we find ourselves working with very complex codebases that make tracking where things come from very difficult. If there is a flow diagram this can help, but often we just need concrete data on where that code goes.
There are a few good ways to do this but the two we are interested in are tracing and callgraphs.

Using Callgraphs

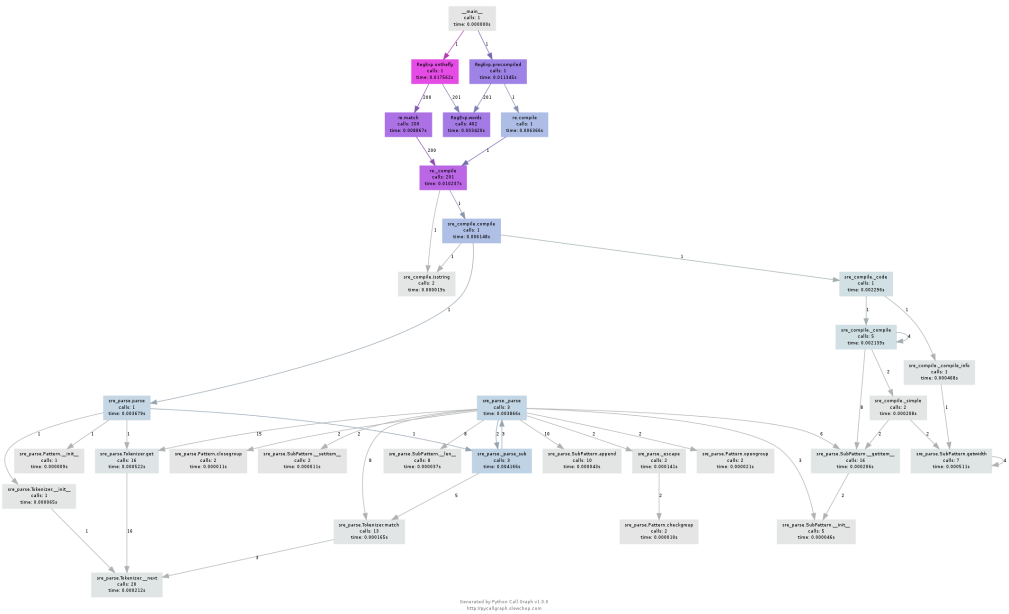
https://pycallgraph.readthedocs.io/en/master/

Callgraphs are a sprawling mess that shows what functions called what during execution, essentially a visual representation of cprofile.
To run one, wrap this around the calling code:

```
from pycallgraph import PyCallGraph
from pycallgraph.output import GraphvizOutput

with PyCallGraph(output=GraphvizOutput(output_file="/path/to/output.png")):
    code_to_profile()
```

Which will output an image like this:



(source – https://pycallgraph.readthedocs.io/en/master/examples/regexp_ungrouped.html)

As you run this you may want to pass a config to reduce how much is displayed, particularly if it's unreadable, this can be limiting depth or using filters to exclude parts of your API you aren't interested in right now.

Using sys.settrace

Sometimes, you just want to follow a specific object as it moves through the callstack, perhaps you have inherited a spaghetti-monster of a codebase and

need to know how data flows through it, or you just need to validate whether the object that makes it to some method is the same as the one you put in.

This is where tracers come in, they are heavy handed and slow (as they get called on every function), but do the job:

```
import sys
ITEMS_TO_TRACE = {}

def tracer(frame, event, arg):
    if event != "call":
        return  # Only check function entry ("return" for exit)
    code = frame.f_code
    if code.co_name == "write":
        return  # print statement

    # used for setting indent, don't query it unless we actually match below.
    depth = -1

    for k, v in frame.f_locals.items():
        k_orig = ITEMS_TO_TRACE.get(id(v))
        if not k_orig:
            continue
        # Match
        if depth == -1:
            depth = 0
            frame_ = frame.f_back
            while frame_:
                frame_ = frame_.f_back
                depth += 1
        s = " " * depth
        s += k_orig
        if k_orig != k:
            s += f"[{k}]"
        s = f"{s: <16}{code.co_name}  {code.co_filename}:{frame.f_lineno}"
        print(s)

a = "hello"
b = "world"
# Set the items to trace
ITEMS_TO_TRACE[id(a)] = "a"
ITEMS_TO_TRACE[id(b)] = "b"
# Enable the tracer
sys.settrace(tracer)
# Some example functions
def func_a(a, b):
    func_b(a, b)
def func_b(c, d):
```

```
    func_c(d, c)
def func_c(a, b):
    pass
func_a(a, b)

# Disable the trace
sys.settrace(None)
```

This will print the below output:

```
 a              func_a  test.py:5
 b              func_a  test.py:5
  a[c]          func_b  test.py:8
  b[d]          func_b  test.py:8
   b[a]         func_c  test.py:10
   a[b]         func_c  test.py:10
```

We can clearly see here what functions are called, where they live, the stack depth and what the variable was renamed to.
Of course there are some limitations, this won't work for compiled code, it won't work for atomic objects like ints and bools, and you'll need to flesh it out further if you want information on non-toplevel objects (eg: if your object is in a list or dict it won't show in the above method).

Inspecting with QEventLoop

Sometimes debugging tools and pipelines goes beyond printing variables or adding breakpoints, particularly in visual applications such as animation and rigging we often need to actually see what is happening part way through a script.
Perhaps there is a node graph or dag structure that only exists for a short time or you need to see baked animation just before the export step.

Fortunately, if you are using a Qt based application such as Maya, Nuke or Houdini you can sneakily leverage QEventLoop to "pause" a running python pipeline, so you can inspect and even change your scene, then let it continue. Sort of like an interactive breakpoint.

```python
from PySide2 import QtWidgets, QtCore

def my_super_long_function():
    print("Running a long function here")
    # Pause the execution
    dialog = QtWidgets.QMessageBox(
        QtWidgets.QMessageBox.Warning,
        "Execution Paused",
        "Click OK to continue")
    # Make sure the dialog doesn't block interaction but stays on top.
    dialog.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
    dialog.setModal(False)
    dialog.show()
    print("Function is paused, have a look around")
    loop = QtCore.QEventLoop()
    dialog.finished.connect(loop.quit)
    # The event loop waits until quit is called,
    # In the meantime, Qt will continue to run the mainevent loop
    loop.exec_()
    print("Function is now continuing")

my_super_long_function()
```

As a small caveat on this, some applications/operating systems may have a different way of setting up Qt dialogs, for example using Houdini on a Mac you need to use the hou.qt module or the application may crash, alternatively you can connect a button or QTimer.singleShot to loop.quit to resume the function.

QObjects

A key part of my developer toolkit when it comes to debugging and modifying existing tools is seeing how they are put together and what is exposed to me. We aren't going to go in depth here, but I did want to briefly touch on what is possible.

Qt has its own metaobject compiler that embeds a ton of useful data into the objects accessible at runtime. Even if the python object is just "QWidget", all the C++ metadata is there for your perusal.
You just need to know where to look.

Attached to every QObject is a QMetaObject (object.metaObject()), this provides access to everything from class attributes and inheritance to methods, enumeration, constructors and properties.
Below is a short overview of all the data you can grab from the metadata:

Figure 2. Qt Metadata

You can use this to hijack properties and values and see what control you have over an object or widget.

I would recommend taking some time to learn more about how Qt stores its metadata. It opens the doors to so many more possibilities for debugging and exploration. For more info, see the documentation. (https://doc.qt.io/qt-5/metaobjects.html)

You can not only view, but edit properties and call methods/signals at runtime. Here is an example script that demonstrates much of the data you can extract from a QObject:

```python
meta_object:QtCore.QMetaObject = qobject.metaObject()
object_name = qobject.objectName()  # This is the exposed name
# Class name whether from C++ or Python, can also use staticMetaObject on a class pointer
class_name = meta_object.className()

# Properties
for name in qobject.dynamicPropertyNames():
    if name == "_PySideInvalidatePtr":
        continue
    try:
        value = qobject.property(name)
    except:  # not all values can be printed
        value = "<unreadable>"
# This is the index of the first property created by this class
property_offset = meta_object.propertyOffset()
# This is the total number of properties on this class
property_count = meta_object.propertyCount()
for i in range(property_offset, property_count):
    # https://doc.qt.io/qt-5/qmetaproperty.html
    meta_property:QtCore.QMetaProperty = meta_object.property(i)
    name:str = meta_property.name()
    try:
        # Note this may error for non-exposed types
        value = meta_property.read(qobject)
    except:
        value = "<unreadable>"

    if meta_property.notifySignalIndex() != -1:
        # this is the signal that is emitted when this property changes
        signal:QtCore.QMetaMethod = meta_property.notifySignal()

# Methods and signals are both stored as methods by Qt
# This is the index of the first method created by this class
method_offset = meta_object.methodOffset()
# This is the total number of methods/signals on this class
method_count = meta_object.methodCount()
for i in range(method_offset, method_count):
    meta_method:QtCore.QMetaMethod = meta_object.method(i)
    signature:str = meta_method.methodSignature()
    match meta_method.methodType():
        case QtCore.QMetaMethod.Signal:
            # It's a signal
        case QtCore.QMetaMethod.Slot:
            # It's a slot
```

And here is an example script to let you click on a widget to find more information about it's hierarchy, properties, signals, slots, etc.
https://gist.github.com/minimaleffortttech/d7aa8fa2ef972db843dcc63f686f2d7f

This is for Nuke, but it's more or less the same code for other DCCs like Maya/Houdini. There isn't much code there, but it should give you an idea as to how you can navigate the Qt metaobject system.

Crashes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Things to look for:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So why do applications crash?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

GDB

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting a stack trace

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Understanding the stack trace

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So what Is the process?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So what should your log look like?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exercise:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Statistics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What not to log

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Profiling

Before you even touch your code, you need to profile to find out what code is running and where the slowdowns actually are.
When running profiles, you need to run several for each scenario to ensure you get a range of numbers to work with.
This gives you a baseline to compare against post-profile as well as a way to validate your changes haven't broken anything.
If using Python, your go-to profiling tool is going to be RunSnakeRun (Linux) or SnakeViz (Windows).
This takes a report from cProfile and gives a visual indication of where time is being consumed.

Important Note

When running profiles, make sure you close your entire python or DCC session between runs.
This is because you will get cached data in future runs.
The only exception to this is when you are **explicitly** profiling the difference between uncached and cached runs or if you are informed the tool is slower each subsequent run.

cProfile

So how do you create a cprofile file?
Simple, you enable it, run your code, disable it and dump the data to a file to open with a viewer.

```
import cProfile

profile = cProfile.Profile()
profile.enable()
# do stuff here, click buttons, whatever
profile.disable()
# print the general stats:
profile.print_stats(sort="time")
# dump to file
profile.dump_stats("/path/to/some/file.profile")
```

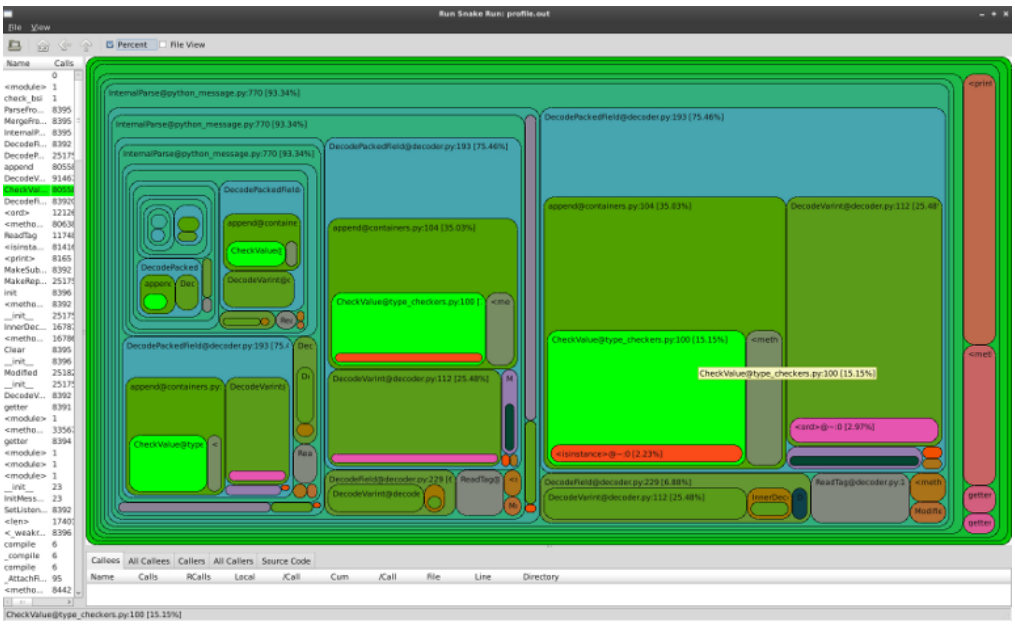


Figure 3. RunSnakeRun

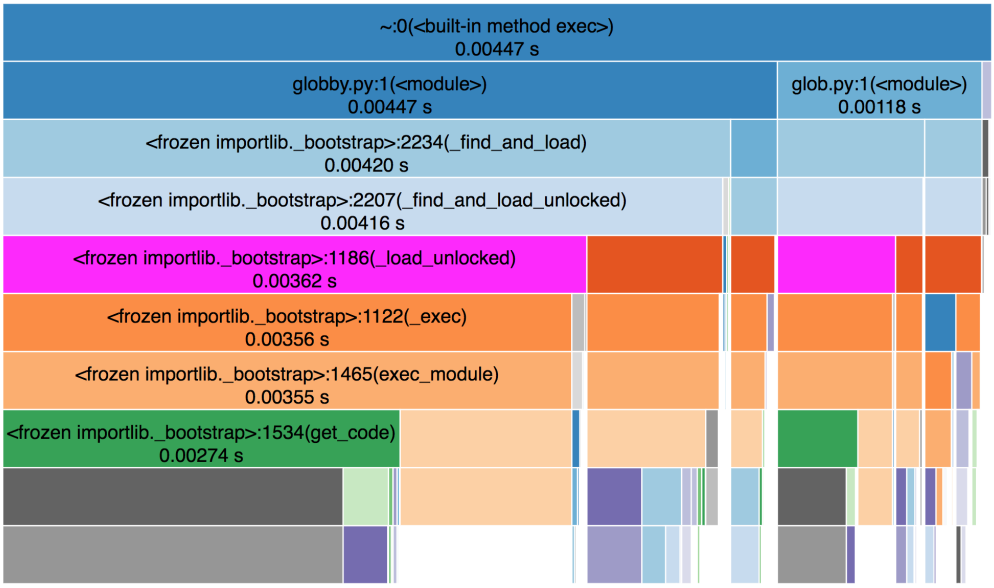


Figure 4. SnakeViz Windows

For RunSnake, the larger the box, the more time it's taken, the larger the empty space, the more time it's spent in that function.
You also want to select a box and see how many cals it's had, the average time of those calls and if it's reentrant/recursive.
For SnakeViz it's the same thing except the boxes are arranged like a traditional profile graph so a bit harder to read at first glance.

Once you have found where you think you are losing performance, take a close look at the stats.
If this function is called a lot, perhaps caching might help, if it's not, then you may want to run a more targeted profile such as line profiler to see whats happening. If it's fast per call but a lot of calls then you may want to see why it is being called so much as you may be able to reduce that count somehow.

Now of course when working with threaded applications you need to spend more time digging because thread waits show as a big box and the calls get dumped all over the place.
If possible, I recommend setting all thread limits to 1 while profiling so as to get a cleaner output. Once you've found functions that need a closer look, here you can use line_profiler.

LineProfiler

https://kernprof.readthedocs.io/en/latest/
Using it is similar to running cProfile:

```
from line_profiler import LineProfiler

profile = LineProfiler()

# Add the functions you are interested in
profile.add_function(func1)
profile.add_function(func2)
profile.add_function(func3)

profile.enable_by_count()
# do things here, click on stuff, whatever.
profile.disable()

# Get the output
profile.print_stats()
```

The output looks something like this:

```
C:\Users\SHUBHAM SAYON\Desktop\Upwork\Ali\progs>python -m line_profiler demo_line_profiler.py.lprof
Timer unit: 1e-06 s

Total time: 10.5219 s
File: demo_line_profiler.py
Function: method_1 at line 5

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     5                                           @profile
     6                                           def method_1():
     7         1   10008648.6 10008648.6     95.1      time.sleep(10)
     8         1     512211.9 512211.9      4.9      a = [random.randint(1, 100) for i in range(100000)]
     9         1       1002.6   1002.6      0.0      res = sum(a) / len(a)
    10         1          0.7      0.7      0.0      return res

Total time: 5.54194 s
File: demo_line_profiler.py
Function: method_2 at line 12

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    12                                           @profile
    13                                           def method_2():
    14         1    5006863.9 5006863.9     90.3      time.sleep(5)
    15         1     534069.2 534069.2      9.6      a = [random.randint(1, 100) for i in range(100000)]
    16         1       1002.6   1002.6      0.0      res = sum(a) / len(a)
    17         1          0.9      0.9      0.0      return res

Total time: 3.56653 s
File: demo_line_profiler.py
Function: method_3 at line 19

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    19                                           @profile
    20                                           def method_3():
    21         1    2995371.4 2995371.4     84.0      time.sleep(3)
    22         1     570151.4 570151.4      16.0      a = [random.randint(1, 100) for i in range(100000)]
    23         1       1003.6   1003.6      0.0      res = sum(a) / len(a)
    24         1          0.7      0.7      0.0      return res
```

Figure 5. Python LineProfiler

Image Source

Like with the cprofile output, we want to look at number of hits, total time and time per hit.
This will show you where in the function we are spending the most time as well as what lines were never used.

MemoryProfiler

Additionally, if you are working with large datasets, you may also want to try memory-profiler:
https://pypi.org/project/memory-profiler/

It's the same result again, but now showing memory allocation and deallocation.
This is useful to see at a glance how large datastructures may be slowing down your application or if you have a memory leak.

Disasembler

Lastly, if you do not have access to the source of the function that is slow and want to take a peek, you can check out the python built in disasembler module. https://docs.python.org/3/library/dis.html

```
>>> dis.dis(myfunc)
  2           RESUME                   0

  3           LOAD_GLOBAL              1 (len + NULL)
              LOAD_FAST                0 (alist)
              CALL                     1
              RETURN_VALUE
```

Figure 6. dis.dis

It's a bit more advanced but if you need to check if a compiled function might be running imports or some other weird business it can give you a glimse. But don't expect to get too much out of this one.

Optimizing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Task 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Threading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Simple Thread Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Simple Async Thread Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

asyncio

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Order of execution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Caveats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Multiprocessing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Deferred Task Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Important notes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What is the subprocess doing?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting information from a subprocess

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting process load

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting open files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Tweaking the environment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Multiprocess Lock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Qt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Overview

Like it or not, Qt is at the heart of many vfx and game productions, it's what powers our DCCs like Maya and Houdini and is often used outside of these too. This is because it is an excellent framework for creating desktop applications and it's free (LGPL).

In this section we are going to cover some of the more advanced aspects of Qt from a production perspective, I will assume you are familiar with PySide or PyQt here.
We will be focusing on QtWidgets as that is the most common Qt library used for desktop applications at this time.
QML is great, but it's still a long way from becoming mainstream.

Qt doesn't belong everywhere

Just because Qt works everywhere, doesn't mean it's necessarily always the best choice.
As our industry develops and we use more and more tools we tend to only use Qt/PySide UIs becuase we don't have to learn new frameworks per tool. Sure Maya, Houdini, Nuke, etc use Qt internally so that "is" the UI framework, but Unreal, Unity, Blender, ZBrush, etc have their own proprietary systems.

The issue with forcing Qt into these tools is that we have to maintain our own event loop and window management, this results in tools that don't quite sync up with the main UI and reduces the overall artist experience. That's not to say you can't use Qt in these tools, just be aware of it's limitations, sometimes Qt can be the sledgehammer solution.

Best Practices

I'd like to take a moment here to go over some ~~pet peeves~~ best practices when working with Qt UIs.

Call down, Emit up, one layer only

Generally speaking, it's ok for a parent object to know about the methods a child object has. It's not ok for a child object to know what methods it's parent has.
This creates a degree of instability in your application and can lead to memory leaks among other issues.
"The underlying C++ object has been deleted" for example.

Good practice is to only call methods on children owned by your object and to do everything else with signals, it does add overhead but it improves decoupling, testability and will likely save a lot of headaches in the long run.
One layer only refers to only interacting with adjacent objects.
For example, calling self.parent().parent().parent().sizeHint() is a sure sign that something has gone horribly wrong somewhere.

Not everything has to be a member variable

I see this everywhere, everything becomes a part of the exposed api, even things that were going to be exposed by Qt anyway:

```
1  self.main_layout = QtWidgets.QVBoxLayout(self)
2  self.label = QtWidgets.QLabel("Label")
3  self.main_layout.addWidget(self.label)
4  self.button = QtWidgets.QPushButton("Button")
5  self.main_layout.addWidget(self.button)
6  self.setLayout(self.main_layout)
```

First, you can access a widgets layout with self.layout(), no need to assign it a variable.
Second, passing self to a layout will assign the layout, you don't need to call self.setLayout.
Third, if you aren't interacting with label and button again, why would you expose them?

Qt will take ownership of the objects you add, you don't need to keep a reference to them.

This is because internally Qt has it's own memory management for an objects lifecycle. These ownership nstructions are communicated with python via the Shiboken typesystem, by holding additional pointers to the objects we run the risk of creating memory leaks (objects that are never cleaned up).

As a quick test you can delete your UI after showing it, if the UI stays open you probably have a variable induced memory leak somewhere. In fact I often include a unit test explicitly for this:

```
import weakref
widget = MyWidget()
widget.show()
ptr = weakref.ref(widget)
del widget
QApplication.processEvents()
assert not ptr
```

Meta Data

An often overlooked part of Qt is the MetaObject.
This is a special object compiled with every class to allow dynamic access to properties, enums, methods and signals.

So why should you care about MetaObjects?
The QMetaObject sits underneath all of your Qt classes to facilitate communication between them, this includes:

- Signals
- Slots/Methods
- Properties
- Class info (Attributes, Inheritance)

There is more it does as well but unless you are doing work with enterprise applications in C++ you can ignore much of the rest.

It is important to know how the data gets to your object though. This is done with the Qt MetaObjectCompiler (MOC), which takes the C++ declarations (or python class definition) and assembles the relevant information into a QMetaObject.

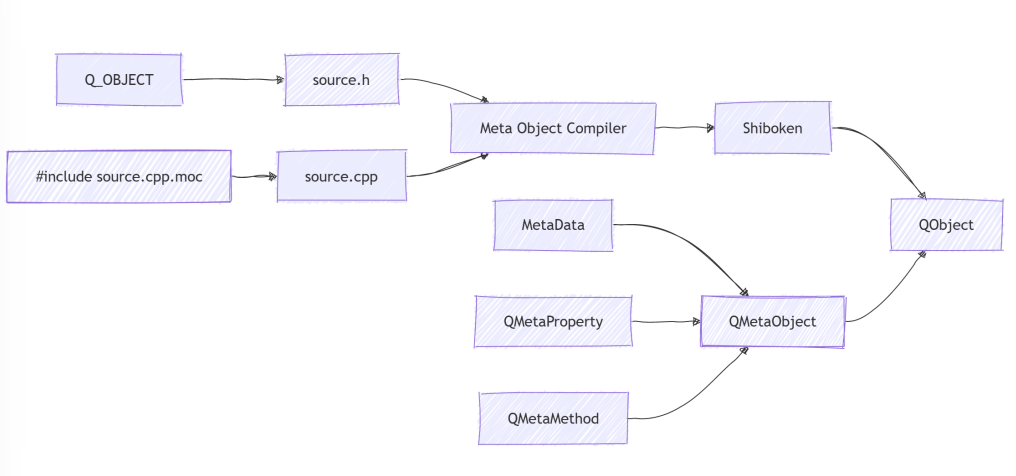


Figure 7. Structure of a QObject

Metadata structure

The QMetaObject contains a lot of information, while I'd recommend reading the official documentation you can see a summary below.
The QMetaMethod provides information about a registered class method (signal or slot).
The QMetaProperty provides information about a registered class property.
These feed into the QMetaObject.

Figure 8. MetaData

Exploring the data

Let's look at a simple example, here is a simple object:
It has custom class info, compiled and dynamic properties, signals and slots. Usually you wouldn't be this verbose in your slot decorators, but we will see how this is helpful shortly.

```
from PySide6 import QtCore

@QtCore.ClassInfo(author="Alex Telford", version="1.0.0")
class MyObject(QtCore.QObject):
    valueChanged = QtCore.Signal(int)
    textChanged = QtCore.Signal(str)

    def __init__(self, parent=None):
        super().__init__(parent)
        self._value = 0
        self._text = ""

        # Set dynamic property
        self.setProperty("objectType", "DemoObject")

    # Slot definitions with result parameter
    @QtCore.Slot(int, result=int)
    def multiplyValue(self, factor):
        result = self._value * factor
        return result
```

```
21
22      # Slot with named parameters
23      @QtCore.Slot(int)
24      def setValue(self, value):
25          if self._value != value:
26              self._value = value
27              self.valueChanged.emit(value)
28
29      @QtCore.Slot(str)
30      def setText(self, text):
31          if self._text != text:
32              self._text = text
33              self.textChanged.emit(text)
34
35      @QtCore.Property(int, notify=valueChanged)
36      def value(self):
37          return self._value
38
39      @value.setter
40      def value(self, val):
41          if self._value != val:
42              self._value = val
43              self.valueChanged.emit(val)
44
45      @QtCore.Property(str, notify=textChanged)
46      def text(self):
47          return self._text
48
49      @text.setter
50      def text(self, txt):
51          if self._text != txt:
52              self._text = txt
53              self.textChanged.emit(txt)
```

What info can we gather from this object?
Before you proceed, I'd like you to write down what you think you will get, then compare.
Here is a function to gather the metadata from your object:

```python
def print_meta_info(obj):
    meta_object = obj.metaObject()
    class_name = meta_object.className()
    object_name = obj.objectName()

    # Print inheritance
    meta_objects = []
    current = meta_object
    while current:
        meta_objects.append(current)
        current = current.superClass()
    inheritance_names = [mo.className() for mo in meta_objects]
    name = f' ({object_name})' if object_name else ''
    print(f"{class_name}{name} {' << '.join(inheritance_names)}")

    # Class Info
    print("\nClass Info:")
    for i in range(meta_object.classInfoCount()):
        class_info = meta_object.classInfo(i)
        name = class_info.name()
        value = class_info.value()
        print(f"- {name} = {value}")

    # Properties
    print("\nDynamic Properties:")
    for prop_name in obj.dynamicPropertyNames():
        prop_name_str = prop_name.data().decode()
        try:
            prop_value = obj.property(prop_name_str)
            print(f"- {prop_name_str} = {prop_value}")
        except:
            print(f"- {prop_name_str} = <unreadable>")

    print("\nProperties:")
    property_offset = meta_object.propertyOffset()

    for i in range(meta_object.propertyCount()):
        prop = meta_object.property(i)
        name = prop.name()
        try:
            value = prop.read(obj)
        except:
            value = "<unreadable>"

        source = "[Local]" if i >= property_offset else "[Inherited]"
        info = f"- {name} = {value} {source}"

        if prop.hasNotifySignal():
            signal = prop.notifySignal()
            signal_signature = signal.methodSignature().data().decode()
```

```
            info += f" notifies: {signal_signature}"

        print(info)

    # Methods and Signals
    signals = []
    slots = []
    method_offset = meta_object.methodOffset()

    for i in range(meta_object.methodCount()):
        method = meta_object.method(i)
        source = "[Local]" if i >= method_offset else "[Inherited]"
        signature = method.methodSignature().data().decode()

        if method.methodType() == QtCore.QMetaMethod.MethodType.Signal:
            signals.append(f"- {signature} {source}")
        elif method.methodType() == QtCore.QMetaMethod.MethodType.Slot:
            return_type = QtCore.QMetaType(method.returnType()).name()
            slots.append(f"- {signature} -> {return_type} {source}")

    print("\nSignals:")
    for signal in signals:
        print(signal)

    print("\nSlots:")
    for slot in slots:
        print(slot)
```

Which results in something like this:

```
MyObject << QObject

Class Info:
- author = Alex Telford
- version = 1.0.0

Dynamic Properties:
- objectType = DemoObject

Properties:
- objectName = '' [Inherited] notifies: objectNameChanged(QString)
- value = 0 [Local] notifies: valueChanged(int)
- text = '' [Local] notifies: textChanged(QString)

Signals:
- destroyed(QObject*) [Inherited]
- destroyed() [Inherited]
```

```
- objectNameChanged(QString) [Inherited]
- valueChanged(int) [Local]
- textChanged(QString) [Local]

Slots:
- deleteLater() -> void [Inherited]
- multiplyValue(int) -> int [Local]
- setValue(int) -> void [Local]
- setText(QString) -> void [Local]
```

We can see what properties we have, signals, slots(methods) and more.

So when would you use this?

There is a lot of information here, but outside of compiling when would this be useful?
Well as it turns out, this is useful in debugging and hacking as well as creating custom decorators.

Not all Qt code is written in python, a lot of it is in C++, specifically, if you are using a DCC like Nuke or Maya then the DCC itself is in C++.
This means that you only have access to the raw QObject and QWidget types. BUT, because the metaobjects remain intact, we can hack into this info for our own purposes.

You can use this to track object lifecycle, "invoke" hidden methods or emit a signal outside the standard process.
Admittedly this is an advanced use case, but it has it's uses, especially in the world of productions where we may need to make sneaky changes to the host UI.

I have an example here of a tool I built for Nuke, it lets you click on a widget to see all the available metadata, it can be easily adapted to another Qt dcc like Nuke or Maya.
https://gist.github.com/minimaleffortech/d7aa8fa2ef972db843dcc63f686f2d7f

Styles and Painting

Qt Styles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

QPalette

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Style Aware Widgets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Color deviations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

QStylePainter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

StyleSheets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

HTML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

State Machines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What are statemachines?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Animation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Named StateMachine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

SCXML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Responsive UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Async

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Ways to async code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What code could be made async?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Qt vs builtin

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Threaded UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Network Requests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Information

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Progress indication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Wait Cursor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Loading Spinner

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Progress Bar

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Status Bar

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Multiprocessing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

State Indication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Adaptive UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Data Driven UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Delegates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Editor Delegate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Painted Delegates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Using your Delegates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Advanced Delegates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Data Mappers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Testing

Just like any other code, UIs need to be tested too. In fact, UI tests can be the most important as by testing them they also test your code in the full context. These are called “Interaction Tests”.

Many TDs don’t write UI tests, often regarded as “too hard” or “too time consuming”.
Personally, I feel this is a mistake. Writing tests in Qt is usually easy and allows you to catch issues that might otherwise go unnoticed.

pytest-qt

Assuming you are using Pytest, you will want to grab the pytest-qt package: https://pytest-qt.readthedocs.io/en/latest/index.html
Technically, you can write Qt tests with C++ or unittest, but it requires a lot more boilerplate setup. Given you can test C++ widgets with pytest-qt as well I find I seldom need to use anything else.

So what does pytest-qt give you?

- qtbot, this is the core of pytest-qt, it manages the lifecycle of widgets, qapplicatoin and simplifies a number of methods.
- Listeners, these wait until a signal or condition is met

There are some other bits around logging but these are the two we will focus on.

Before you test

Before you start writing tests, you need to make your objects retrievable.
Technically, you can add a method that returns the objects so you can test them, but I do not recommend this as it makes them a part of the public API.
Instead, this is where object names come in.

Any non-unique object type you need to test with needs to have an objectName to refer to it by.

I say non-unique because unique objects can be retrieved by their class type (eg: if there is only ever one QSlider, you don't "need" an object name, even if it is good practice).

For example this widget:

```
self._error_label = QtWidgets.QLabel()
self._error_label.setObjectName("error_label")
```

Can be retrieved like this:

```
error_label = widget.findChild(QtWidgets.QLabel, "error_label")
```

There is also findChildren(type) and findChildren(type, name, Qt.FindDirectChildrenOnly) that can be used as needed too.

But wait! can't we just use widget._error_label?
Technically yes, but widgets won't always be exposed as variables, sometimes you need to find them using Qt itself.
An example of this is internal Qt components, such as the OK button in a confirm popup.

The EventLoop

The event loop in Qt (QEventLoop) is what schedules all the actions in your UI, from signals to mouse events to redraws.
When testing, we need to work alongside this event loop becaues certain actions will freeze our python interpreter, such as showing a popup. We do this with waits and timers.

Waits

In pytest-qt, there is some waiters/listeners you can use to wait until a certain condition is met. These are done with context managers.
waitSignal is used to confirm whether a signal was emitted or not and if so what the parameters were:

```python
# Connect a signal to start monitoring
with qtbot.waitSignal(signal_emitter.signal, timeout=1000) as blocker:
    signal_emitter.emit_signal()

# Assert that the signal was emitted
assert blocker.signal_triggered

# Wait for signal with specific arguments
with qtbot.waitSignal(
        signal_emitter.signal,
        timeout=1000,
        check_params_cb=lambda args: args[0] == 'expected') as blocker:
    signal_emitter.emit_signal('expected')
```

waitUntil is used to periodically ping a function until it returns True or it times out

```python
qtbot.waitUntil(lambda: widget.is_ready, timeout=5000)
```

waitExposed is used to wait until a window is shown, this is useful if you have an async UI

```python
qtbot.waitExposed(window)
```

And of course wait will simply wait for a number of msecs:

```python
qtbot.wait(500)
```

Timers

Using timers in Qt allow us to set up an action before we the UI becomes locked. For example a button that shows a popup:

```
QtCore.QTimer.singleShot(20, close_popup)
qtbot.mouseClick(button)
```

You can also use event loops directly if you require it in order to check complex conditions.

```
loop = QtCore.QEventLoop()
myobject.some_signal.connect(loop.quit)
QtCore.QTimer.singleShot(20000, loop.quit)  # 20 seconds
# The python file will freeze here until one of the above conditions are met.
loop.exec_()
# Once we get here the loop is finished either by timeout or some_signal.
```

This is something I use occasionally during debugging tests to simply suspend a test until a timeout or condition is met so I can tweak some things while the test runs.

A Test Widget

For this section, this is the widget we are going to be testing against.
This UI has many of the challenging aspects of Qt testing for us to walk through, such as:

- Testing whether a signal is emitted or not
- Working around popup dialogs
- Checking if an error display shows up
- Interacting with an itemview (combobox)

```
class ParameterWidget(QtWidgets.QWidget):
    parameterSubmitted = Signal(str, str, int)

    def __init__(self, parent=None):
        super().__init__(parent)

        # Create form layout
        main_layout = QtWidgets.QVBoxLayout(self)
        form_layout = QtWidgets.QFormLayout()

        # Line Edit
        self._text_edit = QtWidgets.QLineEdit()
        self._text_edit.setPlaceholderText("Enter text...")

        # Add text edit to form
        form_layout.addRow("Text Input:", self._text_edit)

        # Combobox
        self._combo_box = QtWidgets.QComboBox()
        self._combo_box.addItem("Select...")
        self._combo_box.addItems(
            ["Option 1", "Option 2", "Option 3", "Option 4"])

        # Add combo box to form
        form_layout.addRow("Select Option:", self._combo_box)

        # Slider
        self._slider = QtWidgets.QSlider(Qt.Horizontal)
        self._slider.setRange(0, 100)
        self._slider.setValue(50)

        # Add slider with value to form
        form_layout.addRow("Value:", self._slider)

        main_layout.addLayout(form_layout)

        # Single error message label
        self._error_label = QtWidgets.QLabel()
        self._error_label.setObjectName("error_label")
        self._error_label.setStyleSheet("color: red;")
        self._error_label.setVisible(False)
        main_layout.addWidget(self._error_label)

        main_layout.addStretch(1)

        self._submit_button = QtWidgets.QPushButton("Process Parameters")
        main_layout.addWidget(self._submit_button)

        self._submit_button.clicked.connect(self._on_submit)

```

```
    def _on_submit(self):
        # Reset error state
        self._error_label.setVisible(False)

        # Check if text is empty
        if not self._text_edit.text().strip():
            error_msg = "Text input cannot be empty"
            self._error_label.setText(error_msg)
            self._error_label.setVisible(True)
            return

        # Check if option is selected
        if self._combo_box.currentIndex() == 0:
            error_msg = "Please select a valid option"
            self._error_label.setText(error_msg)
            self._error_label.setVisible(True)
            return

        # If we get here, the form is valid
        text = self._text_edit.text()
        option = self._combo_box.currentText()
        value = self._slider.value()

        self.parameterSubmitted.emit(text, option, value)

        msg_box = QtWidgets.QMessageBox()
        msg_box.setIcon(QtWidgets.QMessageBox.Information)
        msg_box.setWindowTitle("Success")
        msg_box.setText("Parameters submitted successfully!")
        msg_box.setInformativeText(
            f"Text: {text}\nOption: {option}\nValue: {value}")
        msg_box.exec()
```

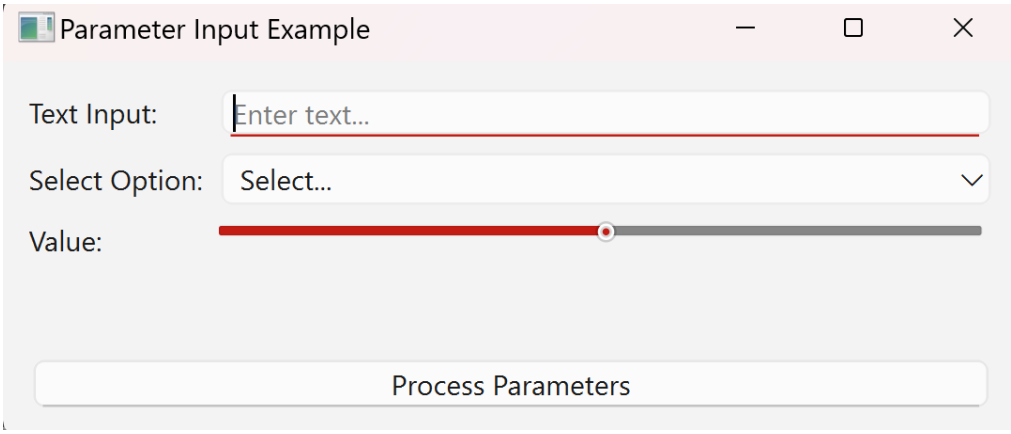


Figure 9. Demo widget

Writing the tests

From here, I will assume you have used pip to install pytest and pytest-qt and are vaguely familiar with writing pytest tests.

Fixtures

To start with, I am going to add a couple of fixtures to help keep the tests short. A fixture is simply a function that gets passed to your tests with optional setup/teardowns.

First, I have a fixture to create our parameter_widget, this shows the widget and adds it to the qtbot registry.
You don't have to add the widget, but this helps ensure they get cleaned up appropriately.

Second, I have a fixture to get the widget references from the UI.
Where possible I like to keep these in one place so if I move or rename a widget I only need to update it once.
I'm also partial to returning dataclasses for convenience, but for now we'll return a tuple.

Note: It is critical that you call assert [widget] on any widget BEFORE you tell qtbot to interact with it, otherwise it will crash without warning.

```python
import pytest
from PySide6 import QtCore, QtWidgets, QtTest
from demo.qt_test import ParameterWidget, process_parameters

@pytest.fixture
def parameter_widget(qtbot):
    widget = ParameterWidget()
    widget.show()
    qtbot.addWidget(widget)
    return widget

@pytest.fixture
def widget_refs(parameter_widget):
    # Find widgets by type
    text_edit = parameter_widget.findChild(QtWidgets.QLineEdit)
    combo_box = parameter_widget.findChild(QtWidgets.QComboBox)
```

```
    slider = parameter_widget.findChild(QtWidgets.QSlider)
    error_label = parameter_widget.findChild(QtWidgets.QLabel, "error_label")

    # Assert widgets exist
    assert text_edit is not None, "Could not find QLineEdit"
    assert combo_box is not None, "Could not find QComboBox"
    assert slider is not None, "Could not find QSlider"
    assert error_label is not None, "Could not find 'error_label'"
    return text_edit, combo_box, slider, error_label
```

Initial State

You don't really need this, but it helps to have a test that checks the default values, this is to ensure that your tests are actually testing things if the defaults change in the future.

```
def test_widget_initial_state(widget_refs):
    """Test the initial state of the widget components."""
    text_edit, combo_box, slider, error_label = widget_refs

    # Check initial text input is empty
    assert text_edit.text() == ""
    assert text_edit.placeholderText() == "Enter text..."

    # Check combobox initial state
    assert combo_box.currentIndex() == 0
    assert combo_box.currentText() == "Select..."
    assert combo_box.count() == 5  # "Select..." + 4 options

    # Check slider initial state
    assert slider.value() == 50
    assert slider.minimum() == 0
    assert slider.maximum() == 100

    # Check error label is initially hidden
    assert not error_label.isVisible()
```

Your first QtTest

Time to actually write some tests, this is fairly straightforward:

1. First, we set the widgets to the desirable state (no text but valid combobox).
   This is because we aren't testing these components here, just the errors.

2. Next we use the qtbot.waitSignal context manager, this will wait for that signal to be emitted or timeout after the code block.
   In this instance, we expect it to not emit so we set raising=False.
3. Verify the expected result.

```python
def test_empty_text_validation(parameter_widget, widget_refs, qtbot):
    """Test validation when text input is empty."""
    text_edit, combo_box, slider, error_label = widget_refs

    text_edit.setText("")
    combo_box.setCurrentIndex(1)
    assert not error_label.isVisible(), "Error label should not be visible yet"
    # Use waitSignal with raising=False to check if signal is emitted
    with qtbot.waitSignal(parameter_widget.parameterSubmitted,
                          raising=False, timeout=100) as signal:
        button = parameter_widget.findChild(QtWidgets.QPushButton)
        qtbot.mouseClick(button, QtCore.Qt.LeftButton)

    # Verify no signal was emitted because validation should fail
    assert not signal.signal_triggered, "Signal with empty text"
    assert error_label.isVisible(), "Error label should be visible"
```

This is a common workflow with Qt tests, we have a lot of waits and event loops to allow Qt to do it's thing after we interact with it, then we check the result.

End-to-end Test

An end-to-end/roundtrip test is when we test our UI from start to finish with an expected result, ideally using interaction only instead of manually setting values.

The reason we try to set things manually is because sometimes widgets will be disabled or hidden, if we are just calling setText or setCurrentIndex then we won't catch these issues.
This test is a little more involved as now we are dealing with signal results, combobox popups and confirm dialogs.
We'll go over the popup functions soon, but for now, here is the workflow:

1. We start by setting our data.

    a. text can be entered with qtbot.keyClicks

b. Comboboxes require popup control so that is handled separately
c. Sliders can be finicky to get the exact point without snapping, for now I have hardcoded this and simply asserted that it is theoretically interactive.

2. As with the last test we wait for the submission signal, but raise is not False as we expect it to emit.

   a. One difference is we have a singleShot to close the confirmation dialog, otherwise it will freeze our tests.

3. Finally, we confirm that the signal was caught and provided the correct arguments.

```
def test_successful_submission(parameter_widget, widget_refs, qtbot):
    """Test successful form submission and signal emission."""
    text_edit, combo_box, slider, error_label = widget_refs

    test_text = "Valid Test Input"
    qtbot.keyClicks(text_edit, test_text)

    select_combobox_item(qtbot, combo_box, 1)  # Select the first real option
    selected_option = combo_box.currentText()

    assert slider.isEnabled()
    assert slider.isVisible()
    slider.setValue(75)
    slider_value = slider.value()

    # Use qtbot.waitSignal to wait for and capture the signal
    with qtbot.waitSignal(parameter_widget.parameterSubmitted) as signal:
        button = parameter_widget.findChild(QtWidgets.QPushButton)
        QtCore.QTimer.singleShot(0, close_popup)
        qtbot.mouseClick(button, QtCore.Qt.LeftButton)

    # Verify signal was emitted with correct values
    assert signal.signal_triggered is True, "Signal should have been triggered"
    assert len(signal.args) == 3, "Signal should have 3 arguments"
    assert signal.args[0] == test_text, f"Expected text '{test_text}', got '{signal.args[\
0]}'"
    assert signal.args[1] == selected_option, f"Expected option '{selected_option}', got \
'{signal.args[1]}'"
    assert signal.args[2] == slider_value, f"Expected value {slider_value}, got {signal.a\
rgs[2]}"
```

Now, let's look into those extra functions.
First, here is the close_popup function:

```
def close_popup(button_text="OK"):
    """Close any open popup dialogs by clicking the confirm button"""
    for widget in QtWidgets.QApplication.topLevelWidgets():
        if isinstance(widget, (QtWidgets.QMessageBox, QtWidgets.QDialog)):
            buttons = widget.findChildren(QtWidgets.QPushButton)
            for button in buttons:
                if button.text().lower() == button_text.lower():
                    QtTest.QTest.mouseClick(button, QtCore.Qt.LeftButton)
                    return True

    if popup := QtWidgets.QApplication.activePopupWidget():
        buttons = popup.findChildren(QtWidgets.QPushButton)
        for button in buttons:
            if button.text().lower() == button_text.lower():
                QtTest.QTest.mouseClick(button, QtCore.Qt.LeftButton)
                return True

    return False
```

This is called from a singleShot as it allows this code to run while the UI is waiting.
It's fairly generic, we iterate over any popup widgets and find a button with the specified text and click it.
Technically you could just close the UI, but I like to be thorough and use interaction as much as possible during a roundtrip.
Next, let's look at the combobox selection, this one is a little more in-depth.

1. First, we check if the index is already the one we want, if so bail early.
2. Next we define a callback function, this is triggered so it will run after we click the combobox button and shows the view.

    a. In this, we confirm the popup showed, we scroll and find the index we require, then we click on it.
    We have to click on the views viewport here for the pseudo-clicks to register.
    b. In the event that the click didn't hide the view, we click on the listview itself to hide it and set the index.

Depending on the platform sometimes the clicks don't work in itemviews.

3. Finally, we trigger the click (which then triggers the above callback), validate it worked and return.

```
def select_combobox_item(qtbot, combo_box, index):
    """Using interaction only, select an item from the combobox."""
    assert combo_box
    if combo_box.currentIndex() == index:
        return

    def callback():
        """Triggered on delay to select the item in the popup view"""
        view = combo_box.view()
        assert view.isVisible()
        model_index = combo_box.model().index(index, 0)
        view.scrollTo(model_index)
        rect = view.visualRect(model_index)
        assert rect.isValid(), "Invalid item rectangle"
        point = rect.center()
        qtbot.mouseMove(view.viewport(), point, delay=20)
        qtbot.mouseClick(view.viewport(), QtCore.Qt.LeftButton, pos=point, delay=5)
        if view.isVisible():
            # On some platforms, the select doesn't work,
            # here we click the main view instead which just hides it.
            qtbot.mouseClick(view, QtCore.Qt.LeftButton, pos=point)
            print(f"Failed to select item {index} in combobox")
            combo_box.setCurrentIndex(index)


    with qtbot.waitSignal(combo_box.currentIndexChanged, timeout=1000) as signal:
        QtCore.QTimer.singleShot(50, callback)
        qtbot.mouseClick(combo_box, QtCore.Qt.LeftButton)

    assert combo_box.currentIndex() == index, f"Expected index {index}, got {combo_box.cu\
rrentIndex()}"
```

Guided Tests

Ok ok, production is in full swing and we don't have time to write extensive tests, or perhaps the backend data changes so often that we simply can't keep up. This is where guided tests comes in. Also known as "Human in the Loop" tests.

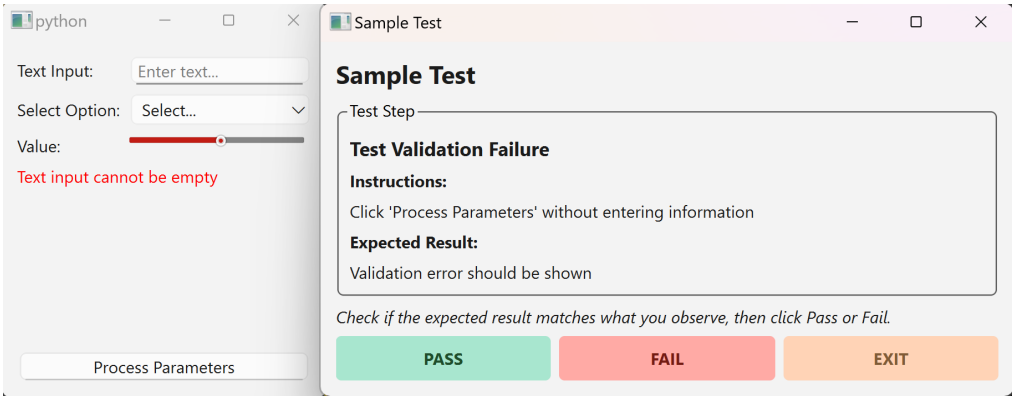


Figure 10. Guided Test

A guided test can work alongside qtbot or you can run it as a standalone thing. Essentially, it is split into a few stages:

1. Pre-setup
   Create the widget, get the UI into the state the user needs to test against.
2. Human-step
   Provide the user instructions to what they need to do, expect some variability here, if you tell the user "Type stuff into the field", they may literally type "stuff" or paste garbage, or potentially something useful.
3. Human-validate
   Ask the user to check if the result matches the expectation, did it show the right message? change to the right panel?
   Sometimes we automate this step.
4. Post-teardown
   This is anything that needs to run to reset the state prior to the next test. This may be nothing or it may be closing the widget and re-opening it, or just resetting the data.

A test case for our UI would look something like this:

```
def test_validation():
    widget = ParameterWidget()
    widget.show()

    guide = GuidedTest("Sample Test")

    with guide.add_step("Test Validation Failure") as step:
        step.set_instructions(
            "Click 'Process Parameters' without entering information")
        step.set_expectation("Validation error should be shown")
        # Do any pre-setup stuff here
        assert step.user_confirm()
        # Do any post-test stuff here

    with guide.add_step("Test Validation Success") as step:
        step.set_instructions(
            "Enter valid info and click 'Process Parameters', then click OK")
        step.set_expectation("Popup should be displayed")
        # Do any pre-setup stuff here
        assert step.user_confirm()
        # Do any post-test stuff here
```

So how does this work?
In essence, we use an event loop.

Create an event loop that waits for the user to click pass/fail, this essentially pauses our test until the condition is met (Just like with pytest-qt).
Below I've included a very simple Guided Test implementation, this will be available in the resources:

```
class GuidedTest(QtWidgets.QWidget):
    def __init__(self, title:str, parent=None):
        super(GuidedTest, self).__init__(parent)
        self.setWindowTitle(title)
        self.setMinimumWidth(500)
        self._event_loop = QtCore.QEventLoop(self)

        layout = QtWidgets.QVBoxLayout(self)

        title_label = QtWidgets.QLabel(title)
        layout.addWidget(title_label)

        # Step panel
        self._step_panel = QtWidgets.QGroupBox("Test Step")
        step_layout = QtWidgets.QVBoxLayout(self._step_panel)

```

```
        self._step_title = QtWidgets.QLabel()
        step_layout.addWidget(self._step_title)

        # Instructions section
        inst_label = QtWidgets.QLabel("Instructions:")
        step_layout.addWidget(inst_label)
        self._instruction_label = QtWidgets.QLabel()
        self._instruction_label.setWordWrap(True)
        step_layout.addWidget(self._instruction_label)

        # Expectations section - initially hidden
        self._expectations_widget = QtWidgets.QWidget()
        expectations_layout = QtWidgets.QVBoxLayout(self._expectations_widget)
        expectations_layout.setContentsMargins(0, 0, 0, 0)

        expect_label = QtWidgets.QLabel("Expected Result:")
        expectations_layout.addWidget(expect_label)
        self._expectation_label = QtWidgets.QLabel()
        self._expectation_label.setWordWrap(True)
        expectations_layout.addWidget(self._expectation_label)

        step_layout.addWidget(self._expectations_widget)
        self._expectations_widget.setVisible(False)

        layout.addWidget(self._step_panel)

        # Info label
        self._info_label = QtWidgets.QLabel(
            "Complete the instructions and click Done.")
        layout.addWidget(self._info_label)

        # Initial "Done" button
        self._done_btn = QtWidgets.QPushButton("Done")
        layout.addWidget(self._done_btn)

        # Pass/Fail buttons container - initially hidden
        self._result_buttons = QtWidgets.QWidget()
        button_layout = QtWidgets.QHBoxLayout(self._result_buttons)
        button_layout.setContentsMargins(0, 0, 0, 0)

        self._pass_btn = QtWidgets.QPushButton("PASS")
        button_layout.addWidget(self._pass_btn)

        self._fail_btn = QtWidgets.QPushButton("FAIL")
        button_layout.addWidget(self._fail_btn)

        self._exit_btn = QtWidgets.QPushButton("EXIT")
        button_layout.addWidget(self._exit_btn)

        layout.addWidget(self._result_buttons)
```

```
        self._result_buttons.setVisible(False)

        # Connect button signals
        self._done_btn.clicked.connect(self._on_done)
        self._pass_btn.clicked.connect(self._on_pass)
        self._fail_btn.clicked.connect(self._on_fail)
        self._exit_btn.clicked.connect(self._on_exit)

        # Initialize state
        self._last_status = False

        self.show()

    def add_step(self, title:str):
        self._step_title.setText(title)

        # Reset UI to instruction phase
        self._expectations_widget.setVisible(False)
        self._result_buttons.setVisible(False)
        self._done_btn.setVisible(True)
        self._info_label.setText("Complete the instructions and click Done.")

        return self

    def set_instructions(self, instructions:str):
        self._instruction_label.setText(instructions)

    def set_expectation(self, expectation:str):
        self._expectation_label.setText(expectation)

    def _on_done(self):
        # Switch to expectation phase
        self._expectations_widget.setVisible(True)
        self._done_btn.setVisible(False)
        self._result_buttons.setVisible(True)
        self._info_label.setText(
            "Check if the expected result matches what you observe, "
            "then click Pass or Fail.")

    def _on_pass(self):
        self._last_status = True
        self._finish_step()

    def _on_fail(self):
        self._last_status = False
        self._finish_step()

    def _on_exit(self):
        self._finish_step()
        raise RuntimeError("Test was manually exited by the user")
```

```
117
118     def closeEvent(self, event):
119         """Handle the window close event by calling _on_exit
120         and ending any active event loop."""
121         self._finish_step()
122         super().closeEvent(event)
123         self._on_exit()
124
125     def _finish_step(self):
126         if  self._event_loop.isRunning():
127             self._event_loop.quit()
128
129     def user_confirm(self, timeout=None):
130         """ Passes control to the user to confirm the test step."""
131         self._event_loop = QtCore.QEventLoop()
132         if timeout:
133             QtCore.QTimer.singleShot(timeout, self._event_loop.quit)
134
135         self._event_loop.exec_()
136         return self._last_status
137
138     # These are just here so we can use the context manager in the test
139     def __enter__(self):
140         return self
141
142     def __exit__(self, exc_type, exc_val, exc_tb):
143         pass
```

In production we would often take this further by adding auto-validations, logging/metrics and more. But for simple cases, the above is perfectly sufficient to create guided user tests.

And that is the gist of it, there is a full test suite in the resources but I do encourage you to explore this further when writing your own tests.
They aren't hard and add another level of robustness to your tooling.

QML in DCCs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Styling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use the fusion style

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Text Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Use Pane as your base component

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Signals and Slots

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Embedding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

QML in QWidget

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

QWidget in QML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Embedding... elsewhere

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

User Experience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What is UX?

UX is User Experience, this encompasses not just how your users interact with your code, but also how they feel when they use it and how well they can accomplish the given task.

> UX is not a pretty UI. It's not color, fonts, style or effects.
> It explores the **Who** and **Why** to better inform the what and how.
> UX is engineered communication and manipulation **for the betterment** of your users.

It's a strong stance, but I say this in an attempt to break the stigma that UX is just pretty designs; it's not.
If your artists are using a User Interface (UI), CommandLine Interface (CLI) or even a physical interface there is a degree of UX required to make that process as streamlined as it can be.

UX does encompass design, it also encompasses communication, psychology, iconography and engineering. User Experience is essentially the engineering of design.

Things like layouts, colors, effects and phrasing are simply techniques we use as a part of the UX design process.
Even documentation and API entrypoints can be considered a part of UX.

UX can be thought of as the pre-production phase of software development, a series of investigations, diagrams, design systems and guides that are put in place before and during the development of a tool in order to deliver a consistent and pleasant experience to the end user.

When we provide artists with a way to interface with our tools to achieve a certain goal, it becomes our responsibility to make that happen in a way that reduces cognitive load (ie: confusion and frustration).
We do that by using techniques such as layouts and effects to communicate to the artist what they need to do, and to a certain extent we may discretely manipulate them towards preferred choices.

Now, did you get triggered a little by the word "manipulate" in my quote above?
It generally has negative connotations but I have used it here with purpose.
There are often times when we need to give artists a nudge towards a certain choice they wouldn't normally choose; eg: actually including "steps to reproduce" on a support ticket.

This might not benefit the artist right now, but we know it will benefit either them or the team as a whole later.
There are times when you do need to include openly manipulative strategies as a UX process to coerce uncooperative artists into playing ball, it sucks to resort to these measures but by being aware of it we can communicate appropriately and openly.

This is where we walk the fine line between good UX and what's known as "Dark UX", ie: manipulation for the *detriment* of your users. For example a 15 minute timer on a shopping cart deal or that ad that somehow always seems to appear just before you click or tap a button.

If you are unsure whether intentional manipulation is good or dark UX, just ask: Who benefits from this?
If it's the user or team, you're probably fine, if it's anyone else like management, marketing or yourself... maybe get a second opinion.

UX is a complex field that overlaps heavily with development, in the following pages we are going to cherry pick some techniques I feel are important for developers to learn, but will not be diving deep into the field.

Specifically we are looking at the UI design portion of UX as it pertains to rapid development. The UX field does encompass a lot more than this: business needs, brand recognition, iconography, marketing, analysis, etc... just as code is a small part of software development.

The 5 elements of UX

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Scope

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Skeleton

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Surface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Research

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

How long should you spend on research

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Where are we coming from?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What are our limitations?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What resources can we use?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

What have others done?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Wireframing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Prototyping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Paper Prototypes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Layouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Order and Importance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Widget Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Don't move things under them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Space and Alignment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

DataViews

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Tips for reducing data display

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Make columns configurable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Save TreeViews for when hierarchy is truly important

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Keep dense editors out of tableviews

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

You don't have to stick to standard view types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Choosing a data view

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Animations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

When to use animations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Accessibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Colour

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Layouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Icons

A part that is often overlooked in VFX is icons. How often do you see a Maya shelf with a consistent look and feel? Even within a single UI, you might have multiple styles of icons.

Figure 11. What do any of these do?

Icons are powerful - they allow your users to see at a glance what something might do or see its state without having to read text.
This allows you to compress UIs further to free up screen real estate.

This often comes under the umbrella term “Iconography”, which also includes things like brand recognition, marketing and more.
But let’s focus on the icons for now.

Consistency is key

When using or creating icons for your application, they need to be consistent in terms of style, size and line weights.
If you are using color in your icons, that needs to be consistent as well.

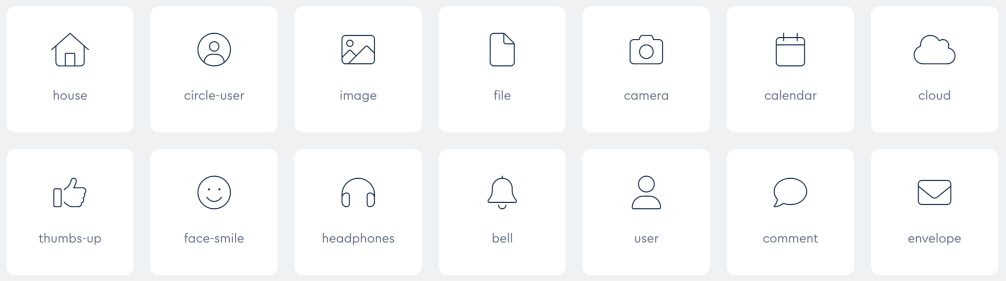


Figure 12. FontAwesome thin icon set

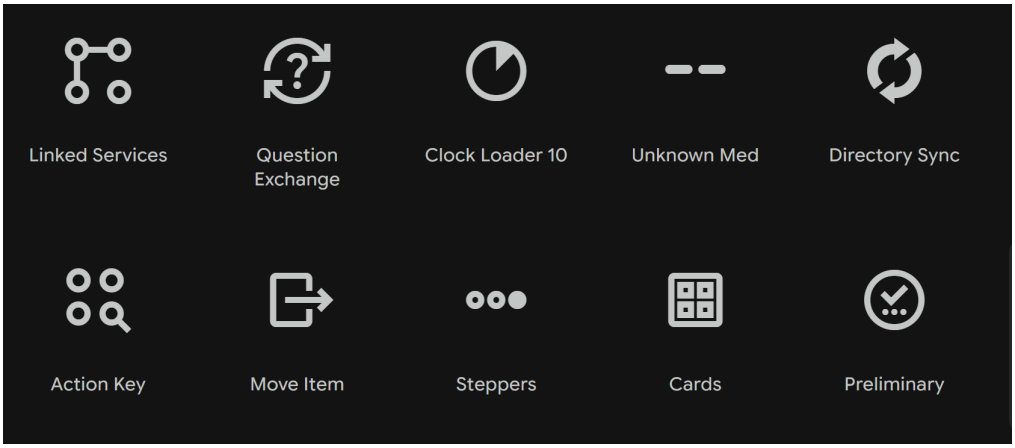


Figure 13. Google Material Icons

Trends come and go, for a while we were all about 3d icons, then it was bold flat icons, then it was thin icons.
Honestly, unless you are selling your software we don't care which trend you use, so long as it's consistent.
If you mix and match 3d, flat solid and thin icons it's not a good look.

Icons should stand on their own

The goal of an icon is to convey what it does to the user, while we usually have them next to related text for context they should ideally be able to stand on their own.

Take these fontawesome icons, most users will know at a glance exactly what they should do:

Figure 14. FontAwesome common icons

In case it's not clear that is "Home", "User/Profile", "Image", "File", "Camera"

Ok, what about these fontawesome icons instead:

Figure 15. FontAwesome uncommon icons

Hmmm, these might need some adjoining text.
You shouldn't worry too much, it's hard to get a perfect and consistent icon without drawing it yourself.
My point here is simply:

> Don't slap an icon on and call it a day, try and find an icon that gives an indication to what the user should expect to happen when they interact with it.

Where to get your icons

I'm going to assume you are not going to be making your own icons, at most you may modify existing ones.
To start with, pick an icon collection that you feel best suits your work and can be reused across many projects.

Unicode

Before you dive into fancy icon packages, your first step should be to check the unicode library.
The benefit of unicode icons is they are directly embeddable into texts without extra fonts or libraries and can be easily colored without custom paint events.
https://www.compart.com/en/unicode/

There are two types of unicode icons, some have no color set, therefore they

inherit from the font color and others have one or more colors embedded into the icon and cannot be colored.
It can be tricky to know which one is which until you use it.
In the example below, I have flat/monochrome icons, some icons with colors embedded and the flat icons again with colors applied.

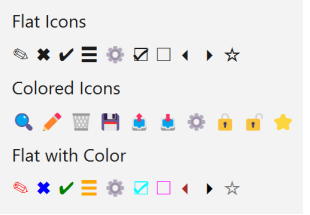


Figure 16. Unicode Icons

What is interesting here is if you look at that cog icon, that is a flat monochrome icon but it still draws gray instead of the purple I requested:

```
colored_label = QtWidgets.QLabel()
rich_text = ""
icon_color_pairs = [
    ("✎", "red"),
    ("✖", "blue"),
    ("✔", "green"),
    ("☰", "orange"),
    ("⚙", "purple"),
    ("☑", "cyan"),
    ("☐", "magenta"),
    ("◂", "brown"),
    ("▸", "black"),
    ("☆", "gray")
]
for icon, color in icon_color_pairs:
    rich_text += f'<span style="color:{color};">{icon}</span> '

colored_label.setText(rich_text)
```

Figure 17. Unicode Icon Code

This is where some inherent variability comes in, the font itself has control over how the icons draw, so you will need to test these accordingly to ensure they draw correctly.
Even then, the benefit of being able to embed an icon into the source code far outweighs this minor inconvenience. There are thousands of icons and often one that more or less suits your needs.

Icon Packages

One of the most popular is FontAwesome.
This is because it comes with a bundled “font” set that allows embedding the

icons into text without needing to use the images directly.
There is over 2000 free icons and over 50,000 pro icons.
The pro icons also can be bought as a one off for a version instead of a subscription.
https://fontawesome.com/search

The next is google material icons.
There is a little over 2500 free icons here
https://fonts.google.com/icons

Then we have Streamline
They have over 200,000 icons
https://www.streamlinehq.com/

Don't choose a service purely based on the number of icons though, sites that offer tens or hundreds of thousands of icons are often filled with garbage repeated icons.

Customization

Despite having many icons to choose from, chances are you won't find the perfect icon for every situation.
This is in part because you need to stick to the same icon set for consistency which reduces your available icon pool.

To customize an icon, find one that is close enough and modify it in a vector software like Inkscape.
It sounds obvious but I often see developers turn to just drawing an icon themselves or getting one from a different style, but this seldom ends well.

Just get it "close enough". This might mean copying the circle icon and replacing the center with another icon, or perhaps just adding a down arrow to the icon.

Legal notice

It is important to note that if you use free icons you may need to include a credit, You can just add a 3rd-party-licenses file or something to that effect. It's only really a concern if you start distributing your software outside your studio.

Don't think too hard

Seriously.
We have as much trouble choosing icons as we do naming things.
Just keep them consistent and make them "close enough", you can always come back and improve them later.

Don't Make Them Think

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

The Mental Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Simplify your UIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Progressive Disclosure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Prioritize Predictability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Guess and Verify

In an ideal world, all data is correct, artists know exactly what they want and nothing changes between it being loaded and updated.

Sadly that is often not the case, this is where we can guess what the user intends and verify that against real data.

An common example of this is autocorrect, the "real data" is the list of available words and we can guess which of those the user is trying to type and suggest some options to them.
In a real sense, this can be applied to inputs like search fields and comboboxes to allow easy filtering.
We do this through one or more of:

- Levenshtein distance/difflib - Checking how close one text is to another.
- Synonym matching/nltk - Allow using similar words in the search.
- LLM vector distance - Slow but powerful method of using AI to determine the intentions from a block of text.

So what does this look like, let's look at some short examples.

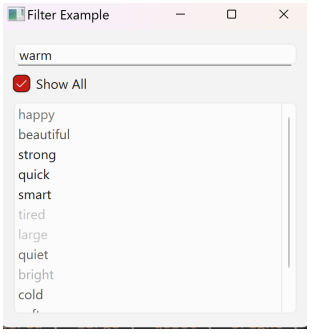


Figure 18. Using difflib to use text distance to account for fumbly fingers and typos.

This is a simple example that demonstrates fumble friendly filtering.
We can compare accuracy against the text itself with optional synonym support:

```python
def simple_match(filter_text, item_text, item_synonyms=None, filter_synonyms=None):
    """Simplified version of the matching algorithm"""
    filter_text = filter_text.lower()
    item_text = item_text.lower()

    # Direct substring match
    if filter_text in item_text:
        return 1.0

    # Direct similarity check
    direct_ratio = SequenceMatcher(None, filter_text, item_text).quick_ratio()

    # Check synonyms if available
    synonym_ratio = 0.0
    if item_synonyms and filter_text:
        for synonym in item_synonyms:
            synonym_ratio = max(
               synonym_ratio,
               SequenceMatcher(None, filter_text, synonym).quick_ratio())

    # Return best match score
    return max(direct_ratio, synonym_ratio)
```

This example simply allows users to mispell and still match the result.
Ideally, users won't even realize they have typed wrong because they immediately get what they want.
In the resources we have a complete Qt filter example for the previous screenshot.

Error handling

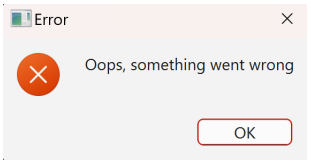


Figure 19. Gee, that's helpful

I can't tell you how many tools I've used that have popups saying "Oh no! something broke", or "An unexpected error occurred, see script editor for details". I may have built my fair share too...

That's not helpful, the user has no clue what happened or what they need to do next.
Error dialogs should have 3 things.

1. A user friendly message explaining what went wrong.
   "The frame range is not valid"
   "Shotgrid service is unavailable"
   "Failed to start application"
2. If not obvious in the error, a message explaining what to do next.
   "Set the end frame after the start frame."
   "Try again in a few minutes or create a ticket if the issue persists."
   "Check you have permissions to [project]"
3. A way to get help.
   This can be a button that shows a ticket submitter or even just a link to some documentation.

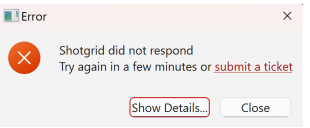


Figure 20. Still broken, but now the user knows what to do next.

Don't Make Them Wait

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Async UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Move queries to a separate thread

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Event Loops for UI Responsiveness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Running in a Separate Process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Inform the user

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Progress Bars

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Loading Spinner

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Wait Cursor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Monitoring External Processes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Assume first, Error later

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Optimistic UI Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Lazy Loading and paging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Adding Redundancy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So what is redundancy protection?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So, what can we do about it?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

The most important redundancy pathway

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So how can we test if our code has enough redundancy?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Further Reading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Books

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Videos

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI in Production Studios

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI Portals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

OpenRouter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Ollama

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Terminology

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

LLM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

ML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Vector Store / Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

PCA

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

RAG

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

MCP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Context Length

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Temperature

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Data Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Choosing an LLM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Memory Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Quantization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Fine Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So where do we start?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Prompt Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Prompt templates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Keep it simple

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Prompt Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

One-Shot, Multi-Shot and No-Shot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Thinking models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Formatting Outputs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Schema

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Catering to a LLM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Prompt Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

How do you stop it?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Running Locally

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Local Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Installing Ollama

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

VLLM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Local AI Apps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Anything LLM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

LM Studio

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

OpenWebUI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Self Hosted Premium Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI Assisted Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Not all agents are the same

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

You are the developer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Slow down

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI Code Editors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Copilot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Disclaimer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Install

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Opt out of training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Chat vs Edit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Edit context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Agent mode

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Extension Settings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Common Instructions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Ollama

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Continue.dev

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Install

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Setting your models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Settings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Common Instructions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Cline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Install

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

plan vs act

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Autonomous agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting the most out of AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Structure

I would recommend adding a few files to each of your projects you want to use with AI:

```
docs/
  style-guide.md
  scratchpad.md
  design-records/
    001-project-structure.md
  diagrams/
  references/
```

It doesn't have to follow this specifically, but this gives a nice blend between modern prompting and traditional engineering.

Style Guide

A style guide is simply a list of instructions to ensure consistency when generating code, this should always be automatically included in your chats if possible (see previous instructions on default prompts). It might look like this for example:

Style Guide

```
## Style Guide for Snake Game PySide6/QML Application

This document outlines the coding standards and style guidelines for the Snake Game proje\
ct. These guidelines should be followed consistently throughout the codebase to maintain
readability and maintainability.

### Python Style Guidelines

#### Naming Conventions
- Use **snake_case** for variable names, function names, and module names
- Use **PascalCase** for class names
- Use **UPPER_SNAKE_CASE** for constants
- Prefix private methods and variables with a single underscore (_)

#### Type Hints
- Always use type hints for function parameters and return values
- Use the typing module for complex types (List, Dict, Optional, etc.)

#### Imports
- Group imports in the following order, separated by a blank line:
  1. Standard library imports
  2. Third-party library imports (including PySide6)
  3. Local application imports
- Sort imports alphabetically within each group

#### Documentation
- Use docstrings for all modules, classes, and functions
- Follow Google style docstrings

#### Code Structure
- Maximum line length: 120 characters
- Use 4 spaces for indentation (no tabs)
- Use vertical whitespace to separate logical sections of code

### Asset Guidelines

#### Icons
- Use SVG format for all icons
- Place all icon files in the `assets/icons` directory
- Do not rename icons if downloaded.

### File Structure
- Place Python modules in the `src` directory
- Place QML files in the `qml` directory
- Organize QML files by feature or component type
- Place asset files (images, fonts, etc.) in the `assets` directory

### Commenting
- Add comments for complex logic or non-obvious behavior only
- Focus on explaining "why" rather than "what" the code is doing
```

```
### General Best Practices
- Prefer composition over inheritance where appropriate
- Keep classes and functions small and focused
- Use descriptive names that convey purpose and intent
```

These should be kept short but convey the minimum requirements for the style. This will probably be copied from project to project but sometimes I'll get the LLM to generate me a new one and I'll refine it from there.

ScratchPad

A ScratchPad is a file that your agent can use to input it's current thoughts and progress.
You can use it by adding something like this to the prompts:

```
Use the scratchpad.md to store your thoughts:
   - Current work focus
   - Recent changes
   - Next steps
   - Active decisions and considerations
And Progress:
   - What works
   - What's left to build
   - Current status
   - Known issues

Update this file as you make changes.
```

This will add a degree of overhead, but is useful when you need to start up a new agent without losing progress.
I will use this if I am performing a large change using AI such as a refactor or update of many files. For general development you can leave this file out.

Design Records

In Engineering we use something called "Architecture Decision Records" (ADRs) These are used to keep track of what features were requested, who requested them and what was decided on.

A Design Record is similar, but a looser format file that is more focused on implementation details of what is planned and what is done for the purpose of guiding an LLM.
Personally I like to include the following in an ADR or Design Record:

- Stakeholders
  Tells me at a glance who requested this when I inevitably go back to refactor it.
  You can also include the initial request and ticket information if you have it.
- File Structure
  It creates a record of what files were potentially created/altered with this change
- Consequences
  I have AI generate this, a list of positives and challenges when implementing this api or feature.
  This acts as a sounding board for things I may not have considered
- Checklist
  A fixed list of actions that we set out to achieve, these start out unticked but as we progress through the changes we tick them off or have the LLM do so.
  This helps to keep the LLM on track.

You can generate your Design Records with an LLM, but always go through it with a fine tooth comb. AI tends to add fanciful features that you didn't specify. I once had it sneak in that I was going to add VR support to a file browser...
A record might look like this:

001-package-structure.md

```
## 1. Package Structure

### Status
Completed

### Context
Design for our File Browser application built with PySide6 and QML that follows best prac\
tices.

### Stakeholders
- Alex Telford (Project Lead, Developer)
- Sam Smith (QA)

### Features and Requirements

#### Core Features
1. File navigation and browsing
2. File operations (copy, move, delete)
3. Search functionality
4. File previews

#### UI Requirements
1. Modern, responsive interface
2. Clear visual feedback
3. Consistent styling

#### Technical Requirements
1. Separation between logic and presentation
2. Maintainable architecture
3. Persistent settings
4. Cross-platform compatibility

### Decision
We will structure the application using the following package organization:

```
file_browser/
├── assets/
│   ├── images/
├── qml/
│   ├── components/
│   │   └── qmldir
│   ├── views/
│   │   ├── MainView.qml
│   │   ├── FileListView.qml
│   │   └── DetailView.qml
│   ├── main.qml
│   └── qmldir
├── src/
│   ├── controllers/
```
```

```
51  │   │   └── file_controller.py
52  │   ├── models/
53  │   │   └── file_model.py
54  │   └── main.py
55  ├── tests/
56  │   ├── unit/
57  │   └── integration/
58  └── requirements.txt
59  ```
60
61  ### Implementation Checklist
62
63  - [✓] Project setup
64    - [✓] Create directory structure
65    - [✓] Set up virtual environment
66    - [✓] Install dependencies
67
68  - [✓] Core Components
69    - [✓] Implement file system model
70    - [✓] Create file operations controller
71    - [✗] Develop search functionality
72
73  - [✓] QML Interface
74    - [✓] Design main layout
75    - [✓] Create file list component
76    - [✓] Implement detail view
77    - [✓] Add navigation controls
78
79  - [✓] Integration
80    - [✓] Connect Python backend with QML frontend
81    - [✓] Implement data binding
82    - [✓] Set up signals and slots
83
84  - [✗] Testing
85    - [✓] Unit tests for core functionality
86    - [✗] Integration tests
87    - [✗] UI testing
88
89  - [✓] Documentation
90    - [✓] Code documentation
91    - [✗] User guide
92    - [✗] Additional ADRs as needed
93
94  ### Consequences
95
96  #### Positive
97  - Clear separation between UI and logic
98  - Modular, extensible structure
99  - Organized asset management
100
```

```
#### Challenges
- Maintaining Python-QML integration
- Ensuring cross-platform compatibility

### Next Steps
1. Implement basic directory structure
2. Set up Python-QML communication
3. Create core file system model
```

That was generated/reviewed, you could also write one like this:

001-package-structure.md

```
## 1. Package Structure

### Stakeholders
- Alex Telford (Lead)
- Sam Smith (QA)

### Task
Create a maintainable structure for the Snake Game using PySide6/QML.

### Architecture Decisions
1. Separate game logic (Python) from presentation (QML)
2. Structure:
   - `src/`: Python backend (game_controller.py, main.py)
   - `qml/`: UI components and screens
   - `assets/`: Images and resources
   - `tests/`: Unit and integration tests
3. Use GameController as central state manager with Qt signals/properties

### Potential Issues
1. QML/Python binding synchronization
2. Performance with property bindings
3. Cross-platform compatibility
```

Entirely up to you, so long as it serves it's purpose, don't overthink it, you don't have to use AI for everything (In fact it's often better you don't).

Diagrams

I love diagrams in code, they are an elegant way to visualize code structure and flow, but they are a pain to make. Fortunately, we can generate these now, they

aren't as good as making them ourselves, but it's quick and often good enough for practical purposes.

The Diagram language of choice for LLMs is "mermaid", this is a syntax that renders in javascript and markdown to an image, meaning it can be embedded in the sidebar.
You can also use PlantUML for complex diagrams but it's a bit hit and miss.

You will need to install these or similar extensions:

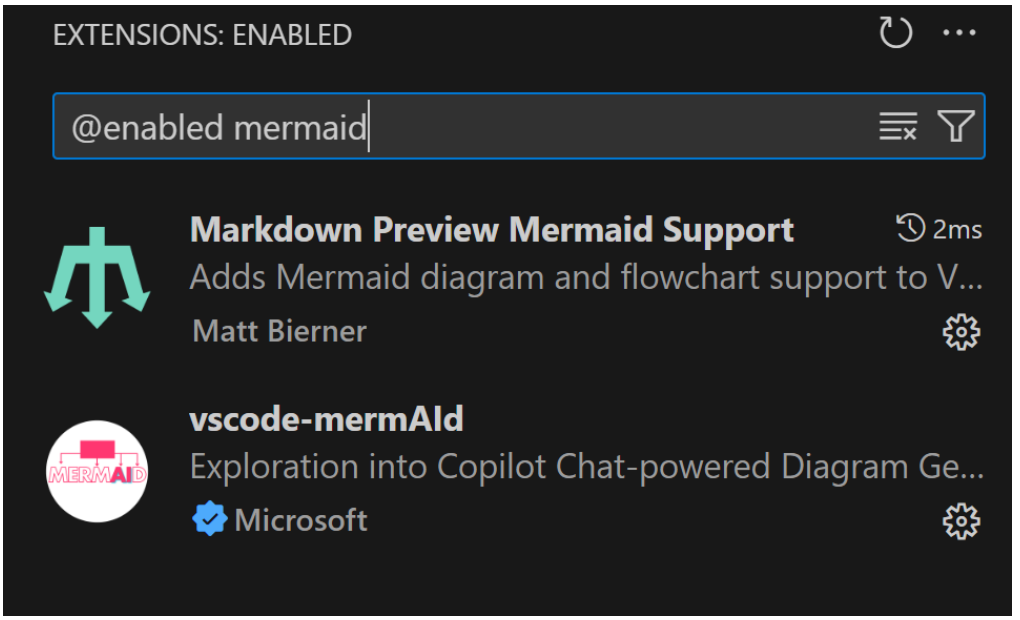


Figure 21. VSCode extensions for Mermaid and PlantUML

Ask either Copilot or Cline to generate a mermaid diagram for what you are trying to build (Or have already built). You can then view this directly in vscode. This is best done once the design records are prepared but before you start development, this ensures that both you and the agent are on the same page in terms of structure and flow.

When requesting a diagram, be explicit about what type of diagram you want and what information you want in it, then iterate until it matches or edit the code directly.

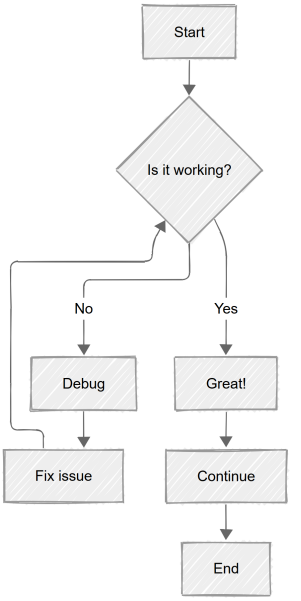


Figure 22. Flow Chart

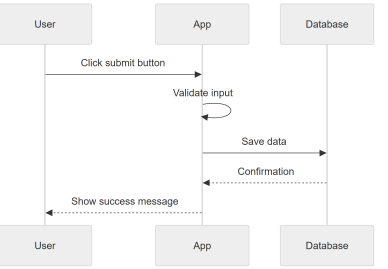


Figure 23. Sequence Diagram

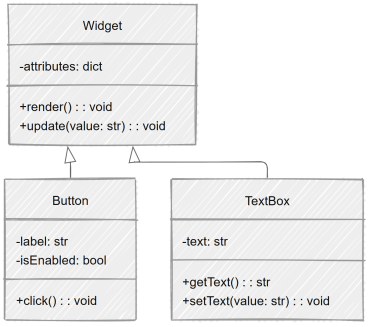


Figure 24. Class Diagram

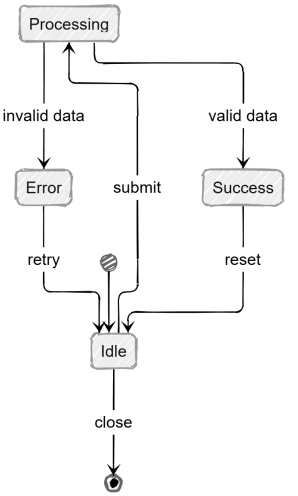


Figure 25. State Diagram

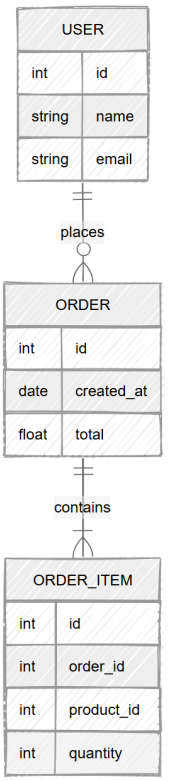


Figure 26. Entity Relationship Diagram

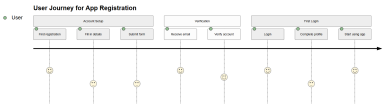


Figure 27. User Journey Diagram

You can mix and match depending on what you are doing, but it's a good idea to generate some up front to make sure you properly understand the problem space, you can even include them in your docs going forward.

Make sure you save your diagrams as markdown not just pngs, this way you can include it in your request to an LLM.

A request would look something like this:

```
Create a user flow chart diagram for my snake app,
use mermaid syntax and save in a markdown file
wrapped in ```mermaid ... ``` tags.

Create a mermaid flowchuser journey diagram for a snake game app. The diagram should have\
 three main sections:

Start Page
Include a title display component
Show the current high score
Add a play button to start the game

Game Page (Main Game Loop)
Begin with game initialization
Create a central game loop with the following components:
A pause check mechanism
Movement handling for the snake
Collision detection system
Food collection check
Score update system
Show how these components flow into each other
Demonstrate how the pause state affects the game loop
Illustrate how collision leads to game over

End Page
Display game over message
Show the final score achieved
Include high score display
```

```
Add logic to check for new high score achievements
Provide a play again button that returns to start

The diagram should use:
Directional arrows to show flow between states
Different colors to highlight key decision points:
Green for positive game actions
Red for game-ending conditions
Yellow for pause functionality
Clear subgraphs to separate the three main game states
Descriptive labels on connection arrows between states
Use the TB (top to bottom) direction for the main flow,
and organize subgraphs to clearly show the progression
from start to game to end states.
```

Which should generate a diagram like this one:

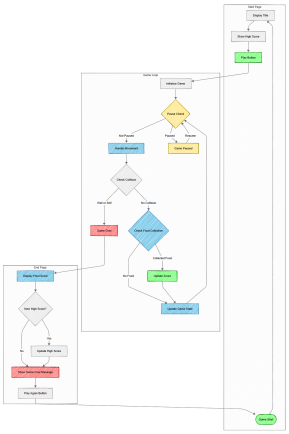


Figure 28. Mermaid user journey diagram for snake game

Depending on the complexity of your project you could look at creating class diagrams, state trees and even timelines.
The key here is that the AI is assisting your development and thought process, not replacing it.

Prompting Code

One thing at a time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Tell it when you made changes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Test, Commit, Rinse and Repeat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Keep the comments in until you are done

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Writing Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Verify, Verify, Verify

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Autonomous Developers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Vibe Coding?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Diagram Generator Case Study

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Getting set up

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Developing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Initial implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Architecture Improvements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Fixing the UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Fixing the Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Build and Deploy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Rewrite Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Rewrite Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Developing AI Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI Structure Recap

At the core of AI tooling is simply a predictive model, usually an LLM.
It consumes text and produces text, we build on this to provide further functionality.
You need to be aware of this structure in order to best understand how to implement your higher level tools.

"**Agents**" are essentially an LLM in a while loop, it will produce text and explanations, and reason what it needs to do next, be it calling a tool or waiting for further instruction,

"**Tools**" are callable functions that are passed in via the LLM context, this will usually be a python function with a name and docstring.

An Agent will use the tool description to decide what tool to use to best achieve it's current task then determine what parameters are valid for that tool before calling it.

"**RAG**" provides a form of embedded knowledge that can be accessed by an LLM to extend the model with knowledge it was not trained on (such as a internal API docs)

"**MCP**" is a protocol standard in which tool collections can be provided to an LLM, for example you could provide "Maya Animation Tools" as an MCP to allow any LLM that supports that standard to interact with Maya animation tools. This is also used to allow multiple different LLM agents to work together

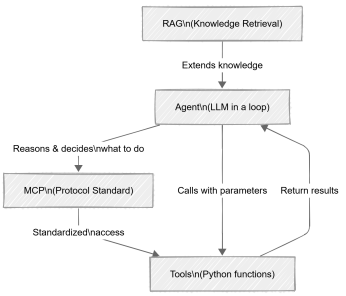


Figure 29. AI tooling structure

Types of AI Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

A simple example

You don't need any APIs to work with LLMs, essentially they are just a networked endpoint.
To illustrate this, let's start with a simple example. Pretend you are creating a tool to help artists submit support tickets. They need to supply:

- A Description of their issue
- Whether it is a bug or feature request
- How urgent it is
- What software it relates to

But artists hate filling in forms.
While we still need to provide the inputs for these options, we can pre-fill them from their description.
Below is a simple example that works with both openrouter and ollama to extract the required info from the text. Replace the model/provider with what you are currently using.
If you missed it in the earlier section, you can download ollama for free and openrouter has a selection of free models (but they will use your data for training purposes).

```
import json
import os
from typing import Optional, List
import requests
from dataclasses import dataclass, asdict, field
from pathlib import Path

@dataclass
class ModelInfo:
    provider: str  # "ollama" or "openrouter"
    model_name: str
    api_key: Optional[str] = None
    api_base: str = None

    def __post_init__(self):
        # Set default API base URL based on provider
        if self.api_base is None:
            if self.provider == "ollama":
                self.api_base = "http://localhost:11434/api"
            elif self.provider == "openrouter":
                self.api_base = "https://openrouter.ai/api/v1"

        # Use environment variable for API key if not provided
        if self.provider == "openrouter" and not self.api_key:
            self.api_key = os.environ.get("OPENROUTER_API_KEY")

@dataclass
class IssueInfo:
    """Information extracted from a user description"""
```

```
    issue_type: str = "unknown"
    sentiment: str = "unknown"
    software: List[str] = field(default_factory=list)

    def model_dump_json(self, indent=None):
        """Compatible replacement for Pydantic's model_dump_json"""
        return json.dumps(asdict(self), indent=indent)

def query_llm(model_info: ModelInfo, prompt: str) -> str:
    if (model_info.provider == "openrouter"):
        if not model_info.api_key:
            raise ValueError("API key is required for OpenRouter")

        response = requests.post(
            f"{model_info.api_base}/chat/completions",
            headers={
                "Authorization": f"Bearer {model_info.api_key}",
                "Content-Type": "application/json"
            },
            json={
                "model": model_info.model_name,
                "messages": [{"role": "user", "content": prompt}]
            }
        )
        response_data = response.json()
        return response_data["choices"][0]["message"]["content"]
    else:  # ollama
        response = requests.post(
            f"{model_info.api_base}/generate",
            json={
                "model": model_info.model_name,
                "prompt": prompt,
                "stream": False
            }
        )
        response_data = response.json()
        if "error" in response_data:
            raise ValueError(f"Error from Ollama: {response_data['error']}")
        return response_data["response"]

def extract_issue_info(
    description: str,
    model_info: ModelInfo
) -> IssueInfo:
    # Create the prompt for the LLM
    prompt = f"""
    Extract the following information from this user description:

    "{description}"

```

```
    Return ONLY a JSON object with these fields:
    {{
        "issue_type": "bug", "feature", or "unknown",
        "sentiment": "distressed", "angry", or "unknown",
        "software": [list of software names] or [] if unknown
    }}
    """
    llm_response = query_llm(model_info, prompt)

    try:
        data = json.loads(llm_response)
    except json.JSONDecodeError:
        # If it fails, try to find JSON in the text
        json_start = llm_response.find('{')
        json_end = llm_response.rfind('}') + 1

        if json_start >= 0 and json_end > json_start:
            json_str = llm_response[json_start:json_end]
            data = json.loads(json_str)
        else:
            return IssueInfo()  # Return default values

    return IssueInfo(
        issue_type=data.get("issue_type", "unknown"),
        sentiment=data.get("sentiment", "unknown"),
        software=data.get("software", [])
    )

if __name__ == "__main__":
    # You can also use openrouter, google/gemma-2-9b-it:free
    model_info = ModelInfo(
        provider="ollama",
        model_name="gemma3:1b",
    )

    inputs = [
        "Maya keeps crashing when I try to open a file. I need it fixed ASAP!",
        "I need to be able to include preroll in publishes.",
        "Nuke is running slow and sometimes freezes.",
        "I don't know what the issue is, but something is wrong with the installation."
    ]
    for test_input in inputs:
        # Using default model (Ollama with llama3)
        result = extract_issue_info(test_input, model_info)
        print("Input:")
        print(test_input)
        print("Result:")
        print(result.model_dump_json(indent=2))
        print("-" * 80)
```

The key aspect here is that the dataclass structure is embedded in the prompt as a json schema.
This helps to ensure the correct data is extracted.

Easy as,
What else can we do with simple LLM tools?
Here is a slightly more in depth example to check out, the task here was to see if we could determine what assets the supervisor notes were referring to and automatically select them for the artist.
https://gist.github.com/minimalefforttech/661e71280096e6d2f3dd0a415fcabc98

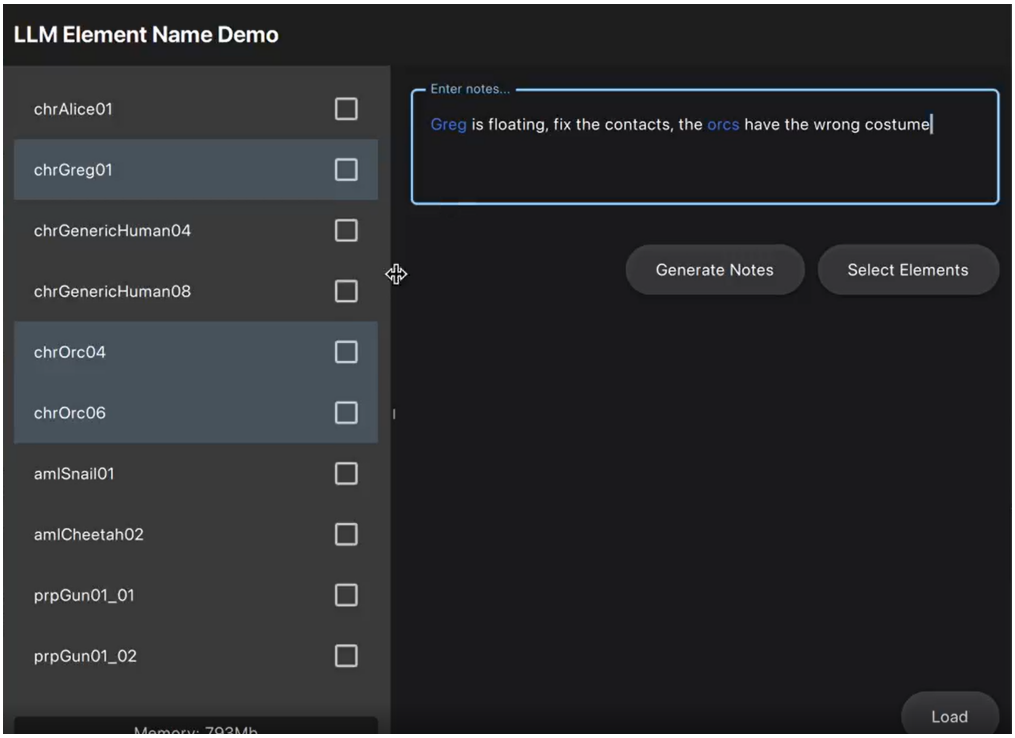


Figure 30. Extract assets from text

A Simple Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Building our agant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Adding RAG vectors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

MCP Servers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Other Important Info

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Legal Licenses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Open Source != Free

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

GPL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Working around the GPL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AGPL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

So who cares?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Further reading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

AI and Machine Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Development Tools and Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Qt Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Package Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Testing and Profiling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

User Experience Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Diagramming and Prototyping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Version Control and Issue Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Production Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Programming Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.

Afterword

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/practical_python.