Quick Guide To Python

Learn Python Programming

Through Real Examples

Asim Jalis

Step-by-Step Guide

Learn Python Programming

Through Real Examples

Asim Jalis

This book is for sale at http://leanpub.com/learnpython

This version was published on 2013-09-18

Tweet This Book!

Please help Asim Jalis by spreading the word about this book on Twitter!

The suggested hashtag for this book is #learnpython.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#learnpython

To my parents who I owe everything.

Contents

Introduction

Instructor

- Asim Jalis
- Has worked as software engineer at Microsoft, Hewlett-Packard, and Salesforce.
- http://linkedin.com/in/asimjalis

Introductions

- What is your name? What do you do?
- How are you planning to use what you learn here?
- What is your perfect outcome?

Hands-On Learning

- How will this course work?
  - Hands-on class.
  - Learn by doing.
- Why hands-on?
  - Helps you get most out of class.
  - You interact with material more deeply, learn.
  - Encourages small mistakes, faster learning.
  - Helps get issues resolved here, now.
  - Afterwards you retain the experience.

History of Python

Guido van Rossum

Who wrote Python?

Python was written by Guido van Rossum, also known as Python’s Benevolent Dictator for Life (BDFL).

What does he look like?

Here[1] is a picture from 2006.

Python’s Goals

What led him to write it?

According to Guido[2], “[In] December 1989, I was looking for a ‘hobby’ programming project that would keep me occupied during the week around Christmas. [...] I decided to write an interpreter for the new scripting language I had been thinking about lately. [...] I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python’s Flying Circus).”

What were some of its goals?

In a 1999 DARPA funding proposal, van Rossum defined Python’s goals as follows:

- an easy and intuitive language just as powerful as major competitors
- open source, so anyone can contribute to its development
- code that is as understandable as plain English
- suitability for everyday tasks, allowing for short development times

Versions and Timeline

What has been Python’s timeline?

[1]http://upload.wikimedia.org/wikipedia/commons/c/c6/Guido_van_Rossum.jpg
[2]http://www.python.org/doc/essays/foreword/

Python Version	Released	Changes
0.9.0	Feb 1991	Released by Guido Van Rossum to alt.sources
1.0	Jan 1994	Added `lambda`, `map`, `filter`, `reduce`
1.5	Dec 1997	Bug fixes
1.6	Sep 2000	License changes
2.0	Oct 2000	List comprehensions
2.1	Apr 2001	Generators
2.2	Dec 2001	
2.3	Jul 2003	
2.4	Nov 2004	
2.5	Sep 2006	
2.6	Oct 2008	
2.7	Jul 2010	
3.0	Dec 2008	Compatibility breaking changes
3.1	Jun 2009	
3.2	Feb 2011	
3.3	Sep 2012	

Python 2.7 Versus Python 3.0

Should I use Python 2.7 or Python 3.0?

You should use Python 2.7 because:

- Many libraries have not been ported over to 3.0
- 2.7 ships with Mac and other systems by default
- You will encounter code written in 2.7

Some of the changes in 3.0 are under the surface and do not affect the syntax. Others do. We will discuss important differences as they come up.

Python 2.7 and 3.0 Differences

Differences in `print`

- Use `print("Hello world")` not `print "Hello world"`
- Use `print("Hello world", end=" ")` not `print "Hello world",`
- Use `print()` not `print`
- Use `print("Error", file=sys.stderr)` not `print>>sys.stderr, "error"`
- Use `print((a, b))` not `print (a,b)`

Differences in integer division

- In 2.7 `3/2` is integer division and so it equals `1`.
- In 3.0 `3/2` is float division and so it equals `1.5`.
- In both 2.7 and 3.0 `3//2` is integer division.
- When you want integer division use `3//2` in both to be safe.

Introduction

What is Python

What is Python?

- Python is a high-level programming language.
- Fast edit-compile-run cycle. Fast, interactive, programming experience.
- Great for writing quick programs, that over time evolve into big programs.
- Useful for automating sysadmin tasks, builds, web sites.
- Ships with Mac and Unix machines by default. Easy to install. Everywhere.
- Object-oriented and the code does not buckle over as it grows over time.

Python vs Perl, Ruby, PHP

All of these languages are scripting languages. The code can be run immediately without waiting for the compiler. This leads to a fast interactive fun programming experience.

Feature	Python	Perl	Ruby	PHP
No compilation needed	X	X	X	X
Web framework	Django, Flask	Catalyst, Dancer	Rails	CakePHP, CodeIgniter
For command-line utilities	X	X	X	
Object-oriented baked-in	X	X		
Scales for large apps	X	X		

Startups Using Python

What are some applications and startups that use Python ?

- BitTorrent[3] was written by Bram Cohen in Python. He started in April 2001, and released it in July 2001.

[3] http://en.wikipedia.org/wiki/BitTorrent_

- eGroups[4] was written in 200,000 lines of Python according to founder Scott Hassan. It was later acquired by Yahoo! and turned into Yahoo! Groups.
- Yelp[5] is written in Python.
- Reddit[6] was written in Lisp and then rewritten in Python in one weekend.

Jobs in Different Languages

How does Python compare with the other languages in jobs?

Here[7] is the relative number of jobs based on data from

Installing Python

Exercise: Install Python.

Solution:

- On Windows install Python 2.7.3 from http://www.python.org/getit/
- On Mac Python 2.7 comes preinstalled.
- Verify that you have `python`.

```
python --version
```

Python Scripts

Exercise: Write a script that prints `Hello, world.`

Solution:

- Save this in a text file called `hello.py` using an editor like Sublime Text, TextWrangler, NotePad++, or TextPad.

[4]http://wiki.python.org/moin/OrganizationsUsingPython

[5]http://engineeringblog.yelp.com/2010/10/mrjob-distributed-computing-for-everybody.html

[6]http://www.aaronsw.com/weblog/rewritingreddit

[7]http://www.indeed.com/trendgraph/jobgraph.png?q=Python%2C+Perl%2C+Ruby%2C+PHP

```
#!/usr/bin/env python

print "Hello, world."
```

- Type `python hello.py` to run it.
- Or type `chmod 755 hello.py` and then type `./hello.py` to run it.

Notes:

- It is conventional to put the `.py` extension on Python scripts. However, it is not required. We could have called the program `hello` instead of `hello.py`.
- Whitespace is significant in Python. The indentation of the statement indicates what block it is in as we will see later. For this program to work, the statement must have zero indentation.
- No semicolon is required to end the statement.
- Print automatically puts a newline after the output.
- If you want to print output without a newline put a comma after it.
- This will not print a newline.

```
#!/usr/bin/env python

print "Hello, world.",
```

Python BAT Files

Exercise: Create a Windows BAT file that prints hello world using Python.

Solution:

- On Windows save this to `file.bat` and then you can run it from the command line.

```
@echo off & python.exe -x "%~f0" %* & goto :EOF

print "Hello, world."
```

- Or you can save Python in `file.py` and then use this `file.bat`.

```
@echo off
python file.py
```

Documentation

Exercise: Find the documentation for the `raw_input` function.

Solution:

- In the Python console type `help(raw_input)`.
- `help` can be used with Python functions and objects.

Interactive Shell

Exercise: Calculate `3 + 4` on the interactive shell.

Solution:

- Type `python`.
- At the Python prompt, enter `3 + 4`.

Notes:

- Note the difference between being in the Python shell and in the terminal shell. It is like *Inception*–shell inside a shell.
- Python has a strong opinion on everything, including how you should exit the shell. To see this try exiting the shell by typing `exit` or `quit`.

Exercise: What is the Zen of Python?

Solution:

- Type `python`.
- At the Python prompt, enter `import this`.

Exercise: Run a script from the Python shell.

Solution: You can run a Python script from the Python shell using `execfile` as follows.

```
execfile(r'/path/to/script/hello.py')
```

Notes:

- Compare the difference between running scripts from the Python shell versus running the script from the terminal or the cmd shell.

Numbers

Integers and Floating-Point Numbers

The most basic data type in Python is the number. Python has both integers as well as floating point numbers. You can enter numbers literally into the shell to see their values.

Exercise: Guess the values and types of these number literals: `123`, `0x10`, `010`, `0`, `1.1`, `1.1e3`, `0.0`

Solution:

Literal	Value	Type
`123`	`123`	integer
`0x10`	`16`	integer
`010`	`8`	integer
`0`	`0`	integer
`1.1`	`1.1`	floating-point
`1.1e3`	`1100.0`	floating-point
`0.0`	`0.0`	floating-point

Variables

Exercise: Write a program that divides a restaurant check of $43 between 3 friends.

Solution:

```python
# Divide restaurant check of $43 between 3 people.

persons = 3
amount = 43.0
amount_per_person = \
    amount / persons
print "Amount per person:", amount_per_person
```

Notes:

- Variables are slots in memory where data such as numbers are stored.
- A variable acquires a value after it is assigned.

- The = operator assigns the value from the right to the variable on its left. Its left hand side can only contain a single variable.
- Variables and numbers can be combined on the right hand side in arithmetic expressions.

Syntactic Notes:

- Statements are executed in sequence.
- Statements end with newline.
- Whitespace and indentation is important.
- Statements must occur on one line.
- If a statement must continue on the following line, end the line with a `\`.
- Comments start with `#` and end at the end of line.
- Python programs are minimal and have very little syntactic noise: no semicolons, no `$` before variable names, no mysterious symbols.

Python Conventions

- Variable names are made up of a letter or underscore, followed by letters, underscores, or numbers.
- Python convention is to use `snake_case` rather than `camelCase` or `PascalCase` for variable and function names.
- Python class names use `PascalCase`.
- Python module names which are the same as file names also use `snake_case`.
- Naming conventions and style guidelines are discussed in more detail in PEP 8 http://www.python.org/dev/peps/pep-0008.
- PEP stands for *Python Enhancement Proposal.*

Arithmetic Operators

What arithmetic operators does Python have?

Expression	Result
`a + b`	Adding `a` and `b`
`a - b`	Subtracting `b` from `a`
`a * b`	Multiplying `a` and `b`
`a / b`	Dividing `a` by `b` (produces fraction in 3.0)
`a // b`	Dividing `a` by `b` (integer division)
`a ** b`	Raising `a` to the power `b`
`a % b`	Remainder of dividing `a` by `b`

Notes:

- These operators don't change the value of the variables in the expression.
- They produce a new value which can be assigned to a variable.
- In `x = a + b` the value of `a` and `b` are not changed. Only the value of `x`. Only the variable to the left of = changes.

Assignment Operators

Exercise: A large 14'' pizza at *Extreme Pizza* costs $14.45. Each topping is $1.70. Suppose we want jalapenos, olives, artichoke hearts, and sun-dried tomatoes. How much will the total be?

Solution 1:

```
crust = 14.45
topping = 1.70

price = crust
price = price + topping    # jalapenos
price = price + topping    # olives
price = price + topping    # artichoke hearts
price = price + topping    # sun-dried tomatoes

print "Pizza Price:", price
```

Notes:

- A variable can be assigned to multiple times.
- A variable can recycle its own previous value on the left of =.
- = is not mathematical equality. So `price = price + topping` is not a paradox.
- = is simply assignment.

Solution 2:

```
crust = 14.45
topping = 1.70

price = crust
price += topping # basil
price += topping # olives
price += topping # cilantro
price += topping # pesto

print "Pizza Price:", price
```

Notes:

- Because the pattern `x = x + a` occurs a lot there is a short-hand for it: `x += a`.
- Read this as: modify `x` by adding `a` to it.
- `+=` is called an assignment operator.
- It is a relative of `=`.
- It too changes the value of the variable only on its left hand side.

What are the different assignment operators?

Assignment	Meaning
`a = b`	Modify `a` by setting it to `b`
`a += b`	Modify `a` by adding `b` to it
`a -= b`	Modify `a` by subtracting `b` from it
`a *= b`	Modify `a` by multiplying `b` with it
`a /= b`	Modify `a` by dividing it by `b`
`a **= b`	Modify `a` by raising it to the power `b`
`a %= b`	Modify `a` by setting it to the remainder of dividing it by `b`

Incrementing and Decrementing

Exercise: Write the statement for incrementing the value of `a` by `1`.

Solution:

```
a += 1
```

Exercise: Write the statement for decrementing the value of `a` by `1`.

Solution:

```
a -= 1
```

Tip Calculator

Exercise: Write a program that divides a restaurant check of $43 between 3 friends and adds a tip as well.

Solution:

```
persons = 3
amount = 43.0
tip_rate = 0.15
tip = amount * tip_rate
amount_per_person = amount / persons
tip_per_person = tip / persons
amount_per_person += tip_per_person
print "Amount per person:", amount_per_person
```

Strings

Strings

Besides integers and floating-point numbers you can also use strings as a data type. String represent text.

Strings are marked by double-quotes or single-quotes. These quotes have the same meaning.

```
message1 = 'Hello'
message2 = 'Goodbye'
print message1
print message2
```

Single-Quotes and Double-Quotes

What's the difference between single-quotes and double-quotes?

- There is no difference in Python.
- You can enclose a single-quote easily in a double-quoted string, and a double-quote in a single-quoted string.
- Escape sequences like `\n` work in both double-quotes, as well as in single-quotes.

What other escape sequences are there?

Sequence	Value
`\n`	Newline
`\t`	Tab
`\a`	Bell
`\'`	Single-quote
`\"`	Double-quote
`\\`	Backslash

Exercise: Print `Jim's Garage`.

Solution 1:

```
print 'Jim\'s Garage'
```

Solution 2:

```
print "Jim's Garage"
```

Raw Strings

Exercise: Print `c:\temp\dir\file.txt`.

Solution 1::

```
print 'c:\\temp\\dir\\file.txt'
```

Solution 2:

```
print r'c:\temp\dir\file.txt'
```

Solution 3:

```
print r"c:\temp\dir\file.txt"
```

Notes:

- The `r` prefix before the string turns it into a raw string.
- What you see is what you get.
- The backslashes are no longer escape characters. Instead they are just backslashes.
- The raw string is like a zip lock bag. The string is preserved exactly as you save it.
- You can put the `r` prefix before a single-quoted string, a double-quoted string, or a triple-quoted string. It has the same effect in all cases.

Triple Quotes

Python also has triple-quotes for strings that span multiple lines.

Exercise: Define `usage_text` for a tip calculator.

Solution:

```
usage_text= '''TIP CALCULATOR

USAGE
    python tip.py AMOUNT PERSONS TIP_RATE

NOTES
    Prints out the total amount due per person including tip.'''
```

Notes:

- The opening `'''` have to be immediately before the first line to avoid a blank line at the beginning.
- The closing `'''` have to be immediately after the last line to avoid a newline at the end.

Concatenating Strings

Strings can be concatenated using the + operator.

Exercise: Combine a first name and last name with a space in the middle.

Solution 1:

```
first_name = 'Dmitri'
last_name = 'Hayward'
full_name = first_name + ' ' + last_name
print full_name
```

Solution 2:

```
name = ''
name += 'Dmitri'
name += ' '
name += 'Hayward'
print name
```

Notes:

- We could have written `name += 'Dmitri'` as `name = name + 'Dmitri'` as well.
- The assignment operator += for strings means modify the variable on the left hand side by appending the expression on the right hand side to it.

Converting Between Numbers and Strings

Exercise: Convert a string `"43.0"` to a floating-point number.

Solution:

```
amount_string = "43.0"
amount = float(amount_string)
```

Exercise: Convert a string `"3"` to an integer.

Solution:

```
people_string = "3"
people = int(people_string)
```

Exercise: Convert a number `14.33` to a string.

Solution:

```
amount_per_person = 14.33
amount_per_person_string = str(14.33)
```

Notes:

- The function `int` converts all data types to integers.
- Similarly, the function `float` converts its input to floating-point number.
- And the function `str` convert any data type to a string.

Exercise: Check what happens if you call `float("hello")`.

String Formatting

Exercise: Write a program that divides a restaurant check of $43 between 3 friends. Print the answer formatted nicely with a dollar sign.

Solution:

```
persons = 3
amount = 43.0
amount_per_person = amount / persons
print "Amount per person: ${0}".format(amount_per_person)
```

Notes:

- The `.format` function is applied to a string and returns a string which is printed by `print`.
- We could have saved it as a variable as well.

```
output = "Amount per person: ${0}".format(amount_per_person)
print output
```

Exercise: Trim the amount to two decimal places.

Solution:

```
persons = 3
amount = 43.0
amount_per_person = amount / persons
print "Amount per person: ${0:.2f}".format(amount_per_person)
```

Exercise: What will be the output of these format strings?

```
print "{0}, {1}, {2}".format("a", "b", "c")
print "{1}, {2}, {0}".format("a", "b", "c")
print "{2}, {2}, {2}".format("a", "b", "c")
```

Solution:

```
a, b, c
b, c, a
c, c, c
```

Notes:

- The number in the parentheses refers to the position of the argument of format.
- You can use an argument as many times as you want. And in any order.

Printf Formatting

Exercise: Write a program that divides a restaurant check of $43 between 3 friends. Print the answer formatted nicely with a dollar sign.

Solution:

```
persons = 3
amount = 43.0
amount_per_person = amount / persons
print "Amount per person: %.2f" % (amount_per_person)
```

Notes:

- The `%` operator is like the `format` function, except it uses traditional C `printf` format syntax.
- This is deprecated in Python 3.0 and `format` is the preferred way of formatting output.
- However, this is commonly used and you will see this in code often.

Reading Input

Exercise: Write a program that asks a user for his name and then says hello to him or her.

Solution:

```
import readline
user_name = raw_input('What is your name? ')
print 'Hello, {0}!'.format(user_name)
```

Notes:

- You can put `import readline` at the top of the file. This line enables Unix readline editing of the input.
- In Python 3.0 `raw_input` has been renamed to `input`.

Population Calculator

Exercise: Estimate the population of California in 2021. The population in 2012 was 38,000,000 and the growth rate was 1%. The population can be estimated as

```
pop = pop_initial * (1 + growth_rate)**years
```

Here `years` is the number of years since 2012, pop_initial is the population in 2012.

Solution:

```
pop_initial = 38 * 1000 * 1000
year_initial = 2012
year_final = 2021
year_count = year_final - year_initial
pop_rate = 0.01
pop_final = pop_initial * ((1 + pop_rate) ** year_count)
print "The population of California in year {0} will be {1:,.0f}".format(
    year_final, pop_final)
```

Notes:

- `{1:,.0f}` prints a floating point number with zero decimal places and with commas.

Exercise: Are the parentheses around `(1 + $pop_rate) ** $year_count` required?

Solution:

- Precedence from higher to lower is: `** * / + -`
- So strictly parentheses are not needed.
- However, it is safer to use parentheses and not guess precendence if it's not clear.

Exercise: Generalize this program to take any year as input.

Functions

Calling Functions

Exercise: Write a program that throws a dice.

Solution 1:

```
import random
dice = random.randint(1,6)
print 'The dice was {0}'.format(dice)
```

Solution 2:

```
from random import *
dice = randint(1,6)
print 'The dice was {0}'.format(dice)
```

Solution 3:

```
import random as r
dice = r.randint(1,6)
print 'The dice was {0}'.format(dice)
```

Notes:

- `import` pulls in functions from other packages.
- Besides built-in functions, all functions and classes require `import`.
- Usually `import` is put at the beginning of the file, but it can occur anywhere before the function call.
- `import random` imports functions and variables into the `random` namespace.
- `from random import *` imports functions and variables into the current namespace.
- `import random as r` imports functions and variables into the namespace `r`. This is useful if you want to abbreviate a long namespace name.

Defining Functions

Exercise: Write a function that takes a name and returns a greeting, and then call it.

Solution:

```python
name = 'Jim'

def greet(name):
    greeting = "Hello, " + name + "!"
    return greeting

print greet(name)
print greet('Dmitri')
print greet('Alice')
print greet('Jim')
```

Notes:

- The indentation is important. Python determines where the function ends based on the indentation.
- The function parameter `name` contains a copy of the value that is passed into the function call. If you modify `name` in the function that will not affect the caller's variable value.
- If a function has no body you can simply put `pass` in it.
- Functions cannot be called before they are defined.

Exercise: Write two functions. The first converts Celsius to Fahrenheit. The second converts Fahrenheit to Celsius.

Solution:

```python
def c2f(c):
    f = (c * 9./5.) + 32.
    return f

def f2c(f):
    c = (f - 32.) * 5./9.
    return c
```

Notes:

- All variables introduced in the function for example `f` and `c` disappear as soon as the function exits.
- They do not have any connection to variables of the same name outside the function.

Default Argument Values

Exercise: Create a tip calculator function that assumes a default tip rate of 15%.

Solution:

```
def calculate_tip(amount, tip_rate = 0.15):
    tip = amount * float(tip_rate)
    return tip

print calculate_tip(10, 0.20)
print calculate_tip(10, 0.15)
print calculate_tip(10)
```

Keyword Arguments

Exercise: Create a tip calculator function that calculates the total amount per person including tip. It assumes a default tip rate of 15% and assumes that by default there is only one person.

Solution:

```
def calculate_tip(amount, persons=1, tip_rate=0.15):
    tip = amount * float(tip_rate)
    amount += tip
    amount_per_person = amount / persons
    return amount_per_person

print calculate_tip(43, 3, 0.15)
print calculate_tip(43, persons=3, tip_rate=0.20)
print calculate_tip(amount=43, persons=3, tip_rate=0.20)
print calculate_tip(amount=43, tip_rate=0.20)
print calculate_tip(amount=43, tip_rate=0.20, persons=3)
print calculate_tip(tip_rate=0.20, persons=3, amount=43)
```

Notes:

- You can combine position arguments and keyword arguments.
- However, once you start using keywords arguments you cannot go back to positional arguments in that function call.

Multiple Return Values

Exercise: Write a function that prompts the user for amount, people, and tip rate, and then returns all three values.

Solution 1:

```python
def get_input():
    amount = float(raw_input('Amount: '))
    people = int(raw_input('People: '))
    tip_rate = float(raw_input('Tip Rate: '))
    return amount, people, tip_rate

amount, people, tip_rate = get_input()

print amount, people, tip_rate
```

Solution 2:

```python
def get_input():
    amount = float(raw_input('Amount: '))
    people = int(raw_input('People: '))
    tip_rate = float(raw_input('Tip Rate: '))
    return (amount, people, tip_rate)

(amount, people, tip_rate) = get_input()

print amount, people, tip_rate
```

Notes:

- In effect the two solutions are equivalent.
- In both cases the three variables `amount`, `people`, and `tip_rate` will get the values the user entered.
- Under the hood Solution 1 returns a tuple. Solution 2 makes that explicit by using the tuple notation. We will discuss tuples in a future section.
- Solution 1 is more idiomatic and is commonly used in Python code.

Scoping Rules

Python's scoping rules can be remembered using this mneumonic: LEGB.

Scope	Meaning
Local	Defined in the current `def`
Enclosed	Defined in the enclosing `def`
Global	Defined in the module
Builtin	Python builtin

Note that this means that you can override builtin variables.

Exercise: Redefine `raw_input` to throw an exception.

Solution:

```python
def raw_input(message):
    raise Exception(
        "raw_input: not implemented")

name = raw_input("What is your name? ")
print "Hello " + name + "!"
```

Notes:

- When a variable is read it is looked up first in the local scope, and then in enclosed, then global, then builtin. If it is not found in any of them an error is raised.
- When a variable is assigned to it becomes attached to the scope where the assignment happens.
- A more local variable can hide a less local variable within its scope.
- New scopes are introduced in function definitions.
- In general a function can read the variables defined in outer scopes, but cannot write to them.

Globals

Exercise: Define a module variable using `VERBOSE = True`. Then reset it to `False` in a function called `disableVerbose()`.

Solution:

```python
VERBOSE = True

def disableVerbose():
    global VERBOSE
    VERBOSE = False

disableVerbose()
print 'Verbose: ' + str(VERBOSE)
```

Notes:

- The `global` keyword puts variables in the module or global scope within the scope in which it is used.
- Without the `global` keyword the `VERBOSE` inside `disableVerbose` will be redefined as a local variable, and setting it to `False` will have no effect on the `VERBOSE` in the global scope.
- If `global` is used in the module or global scope it has no effect.

Creating Libraries of Functions

Exercise: Create a `util.py` file and then import it into your program.

Solution:

In `util.py`, type:

```python
def greet():
    print "Hello, world"
```

In your Python file type:

```python
import util
util.greet()
```

Notes:

If the file is not in the same directory add these lines first.

```python
import sys
dir = '/path/to/dir-containing-file'
sys.path.append(dir)
```

Objects

Interacting with Objects

How do you interact with objects?

- In Python everything is an object.
- Here is how you interact with objects.

```
obj.method(arg1, arg2, arg3)
```

- Every object has methods on it, which are like functions, except they are specialized to that object.
- For example, a car and a computer both have a *start* function, but they mean specific things for that object.

Why is the point of all this?

- Objects allow the code to be modular and more organized.
- Each part of the system only knows about itself.
- Objects are introverts.
- So changes to an object's insides are less likely to break the system.

Exercise: Look at the string functions on the Python console and then find a way to uppercase a string. You can find the functions for an object using the `dir(obj)` function.

Exercise: Find out if a string ends with a question mark.

Exercise: Find the `format` function on the string.