SimPy: Simulation in Python

Afonso C. Medina

SimPy: Simulation in Python

Afonso C. Medina

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

This version was published on 2025-12-01

Contents

See me, Feel me, README **1**

Preface **3**
- Why use SimPy? 3
- Pros and cons 4
- A brief history of SimPy 5
- Where to find help about SimPy 5
- How to use this book 6

Installing SimPy **8**
- Step 1: Anaconda, the easy way 8
- Step 2: Installing Pip (for those who did not install Anaconda) 9
- Step 3: Installing SimPy 10
- Step 4: Installing an Integrated Development Environment (IDE) 11

It All Depends on Python **13**
- Test Your Python Knowledge: the gambler's ruin problem 13
- Challenges 13
- Solution to Challenge 1 15
- Test Your Knowledge 16

First Steps in SimPy: creating entities **18**
- Importing the random and simpy libraries 18
- Creating a simulation environment 18
- Creating an arrival generator within the environment 19
- Creating time intervals with env.timeout(wait_time) 20
- Running the model for a determined time with env.run(until=simulation_-time) 21
- Concepts from this section 24
- Challenges 24
- Solution to Challenges 2 and 3 25

Test Your Knowledge 28

Creating, occupying and releasing resources 29
Creating 29
Occupying 29
Releasing 29
Resource status 29
Concepts from this section 30

Putting it all together in an example: the M/M/1 queue 31
Entity arrival generation 31
Performing service at the server 31
An alternative representation for occupying and releasing resources . 31
Concepts from this section 31
Challenges 31
Solution to Challenges 4, 5 and 6 32
Test Your Knowledge 32

Attributes and variables: differences in SimPy 33
Attributes in object-oriented models 33
Concepts from this section 33
Challenges 33
Solution to Challenges 7 and 8 34
Test Your Knowledge 34

Environments: controlling the simulation 35
Execution control with env.run(until=end_of_simulation) 35
Stop by execution of all scheduled processes 35
Stop by end of specific process execution with env.run(until=process) 35
Step-by-step simulation: peek & step 35
Concepts from this section 35
Challenges 35
Solution to Challenges 9 and 10 37

Other types of resources: with priority and preemptive 38
Resources with priority: PriorityResource 38
Resources that can be interrupted: PreemptiveResource 38
Concepts from this section 38
Challenges 38
Solution to Challenges 11 and 12 39

Process interruptions: simpy.Interrupt 40
Creating equipment breakdowns 40
Interrupting a process without capture by try...except 40
Concepts from this section 40
Challenges 40
Solution to Challenges 13 and 14 41
Test Your Knowledge 41

Storage and selection of specific objects with Store, FilterStore and PriorityStore 42
Building a set of objects with Store 42
Selecting a specific object with FilterStore() 42
Creating a Store with priority: PriorityStore 42
Concepts from this section 42
Challenges 42
Solution to Challenges 15 and 16 43
Test Your Knowledge 43

Filling or emptying boxes, tanks, stocks or objects with Container() . . 44
Filling my container yield myContainer.put(quantity) 44
Emptying my container: yield myContainer.get(quantity) 44
Creating a sensor for the current container level 44
Concepts from this section 44
Challenges 44
Solution to Challenges 17 and 18 45
Test Your Knowledge 45

Creating batches (or grouping) entities during simulation 46
A tactic for batch grouping using Container 46
Grouping batches by entity attribute using FilterStore 46
Challenges 46
Solution to Challenges 19 and 20 47
Test Your Knowledge 47

Creating, manipulating and triggering events with event() 48
Creating an isolated event with event() 48
Concepts from this section 48
Challenges 48
Solution to Challenges 21 and 22 49
Test Your Knowledge 49

Waiting for multiple events at the same time with AnyOf and AllOf . . . 50
Waiting until at least one event finishes with AnyOf 50
Waiting for all events with AllOf 50
Better understanding the outputs of AllOf and AnyOf commands . . . 50
Concepts from this section 50
Challenges 50
Solution to Challenges 23 and 24 51
Test Your Knowledge 51

Useful properties of events 52
Concepts from this section 52
Challenges 52
Solution to Challenge 25 53
Test Your Knowledge 53

Adding callbacks to events 54
Every process is an event 54
Concepts from this section 54
Challenges 54
Solution to Challenge 26 55
Test Your Knowledge 55

Event interruptions 56
Interrupting an event with the interrupt method 56
Interrupting an event with the fail method 56

What are generator functions? (or how SimPy works) - Part I 57
Iterator 57
Generator functions 57

What are generator functions? (or how SimPy works?) - Part II 58
SimPy vs. generator functions 58

Agent-Based Simulation using SimPy 59
How to build an agent simulation model: basic steps 59
Agent-based epidemic model: the SIR model 59
Modeling the problem in SimPy 59
Improving simulation code performance 60
How to proceed from here 61
Concepts from this section 62

Challenges 62
Solution to Challenges 27 and 28 63
Test Your Knowledge 63

Data input and output via spreadsheet 64
Communication with the library 64

See me, Feel me, README

This book was conceived as an introduction to the **SimPy** library (*Simulation in Python*) for building discrete event simulation models in Python. It is the product of the work of someone who researches, teaches and professionally uses simulation tools in their daily work.

This is a work *in development* on the Leanpub platform. This means that the material is updated periodically, whether for corrections, improvements or additions of new content.

Intentionally, the book is divided into short sections, emphasizing one concept at a time, facilitating the learning process and quick reference material searches.

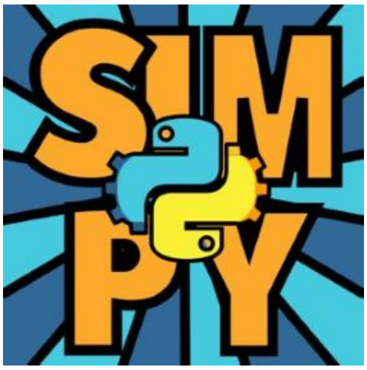


Figure 1. "PsychedelicSimpy!"

Before proceeding, a small notice:

> The work assumes that the reader has knowledge of Discrete Event Modeling and Simulation, is taking a course on the subject, or is reading the prestigious book Modeling and Simulation of Discrete Events (Chwif & Medina).

Therefore, the book is not intended as a reference source on systems modeling methodologies, but rather, a book about one of the most fascinating languages available for building simulation models.

To the dear reader who begins their navigation here, a suggestion: visualize and reflect on a simulation project you would like to develop in SimPy and, with each section of the book completed, try to develop your own project in parallel in harmony with the learned content.

Bon voyage!

Afonso C. Medina

* * *

UPDATE

Current version: 02/2025

What's new:

- Text corrections
- Machine and laundry figures

In development:

- Optimization
- Output statistics

To do:

- AI applications
- Communication with spreadsheets/txt
- Final Code Review
- Final Text Review

Preface

This book was written with compact sections in short texts, so that the reader, at each section, has a clear view of the content presented, building knowledge in a solid way and respecting their personal learning curve.

SimPy (*Simulation in Python*) is a framework for building discrete event simulation models in Python, distributed under the MIT license. It differs from commercial packages usually used in discrete event simulation, as it is not an application with ready-made objects that can be easily connected by simple mouse clicks. With **SimPy**, it is up to the user to build a Python computer program that represents their model. Essentially, **SimPy** is a command library that gives Python the power to build discrete event models.

With **SimPy**, you can build, in addition to discrete simulation models, real-time simulation models, agent-based models, and even continuous simulation models. In fact, as the reader will notice throughout this text, these possibilities are more associated with Python than with the resources provided by **SimPy** itself.

Why use SimPy?

Perhaps the correct question is: "why use Python?"

Python is today, perhaps, the most widely used language in the scientific community, and a brief search on the Internet will suggest articles, posts, and endless discussions about the reasons for all this success. I would summarize Python's success in 3 major reasons:

- **Ease of coding**. Engineers, mathematicians, and researchers in general want to think about the problem, not so much about the language, and Python delivers on its promise when it comes to ease. If it is easy to code, it is even easier to read and interpret code written in Python;
- **Libraries! Libraries!** An unbelievable number of libraries (particularly for the scientific field) is available to the programmer (and researcher).
- **Scripts**. The functionality of working with scripts[1] or small pieces of

[1]A *script* is a sequence of commands executed inside some file through an interpreter. It is called a *script* because they are read and interpreted by Python, line by line.

interpreted code (basically, Python is a "scripting language") drastically reduces development time and language learning.

Furthermore, SimPy, when compared to commercial packages, is free–which in itself is a great advantage in a market where software is priced starting from thousands of dollars–and quite flexible, in the sense that it is not limited only to existing modules.

From a functional standpoint, SimPy presents itself as a Python *library*, and this means that a simulation model developed with it will have at its disposal everything good that exists for those who program in Python: the code is easy to read (and develop), the model can be distributed as a package (without the need for the end user to install Python to run it), in addition to the various statistics and optimization libraries available in Python, which greatly expand the application horizon of the models.

This availability of libraries, as well as being free software, makes SimPy particularly interesting for those who are developing their academic research in the area of simulation. Your model will likely be better documented and therefore easier to understand, enhancing the dissemination of your research results in dissertations, conferences, and scientific articles.

Pros and cons

Pros:

- Open source and free (MIT license);
- Various optimization, mathematical, and statistical library functions can be incorporated into the model;
- Allows programming of sophisticated logic, relying on Python (and its libraries);
- Active community of developers and users that keeps the library up to date.

Cons:

- Performance: although Python's performance improvement with each new version is notable, SimPy is a simulation library, that is: it belongs

to a class of applications that is quite demanding in computational terms. So, don't count on processing times comparable to those of commercial packages, such as AnyLogic or Simul8, because it won't be;
- Lack of animation tools;
- Need to program each process of the model;
- Requires prior knowledge of Python;
- Does not include a visual development environment.

A brief history of SimPy

SimPy emerged in 2002 based on the combination of ideas from Simscript and the Simula family of languages. In context, it combines SiPy, a package developed by Klaus Muller based on Simula, with SimPy, developed by Tony Vignaux and Chang Chui and based on another language, Simscript.

Perhaps what best symbolizes the genesis of SimPy is Prof. Klaus Muller's decision to use *generators* or *generator functions* in building the SiPy package. When the two original projects were merged into SimPy as we know it today, the trio decided to follow the path of *generators*, until then a welcome novelty in Python 2.2 (we are still in 2002, folks!).

In its infancy, SimPy was used in introductory simulation courses and, gradually, it began to attract the interest of researchers and professionals around the world. Starting from version 3.0, SimPy, already counting with a larger number of developers, was completely rewritten and its importance in the simulation world can be measured by the countless papers published in specialized conferences and journals, as well as its reimplementation in other languages, such as: C# (Sim#), Julia (ConcurrentSim), and R (simmer).

Where to find help about SimPy

There is a variety of videos and tutorials available on the Internet, but three are the reference sources I use most:

- The project's own website http://simpy.readthedocs.io, with examples and a detailed description of the Application Programming Interface (API);
- The user mailing list is quite active, with well-crafted answers;

- Stack Overflow has a reasonable number of questions and examples, but be careful, as much of the material still refers to the old version 2 of SimPy, quite different from version 3, the basis of this work.

How to use this book

This text was planned in a compact section format, so that they are short and didactic–with the goal that each section does not go much beyond 500 words. Each chapter presents a new topic through one or more practical examples. Following that, complementary challenges are proposed so that the reader advances their knowledge by practicing on the proposed example itself.

Like every book that introduces a new programming language, the ideal is for the reader to *practice the examples by rewriting the code*. Copying and pasting ready-made code from the text will make you a Jedi master at using the keyboard and mouse for copying and pasting texts, but it won't teach you to program in Python or SimPy. So, my suggestion is that the reader reads each example presented and rewrites it again. Typing errors, unexpected *bugs*, and doubts will arise that, if overcome in the code review, will indeed make the reader a Jedi master of simulation in Python.

This is the first edition of a book about a language that has been attracting increasing interest in recent years. Naturally, the increase in users–and it is hoped that this book will contribute to this–will also cause the emergence of more knowledge, more creative solutions that should be incorporated into this book in future revisions. In fact, this text is already an evolution of the material I made available for some years as a tutorial on gitBook.

Found an error or want to send a suggestion for improvement? Please send an email to: livrosimulacao@gmail.com.

The plan proposed by this text is to follow this sequence (**under permanent revision**):

- Introduction to SimPy;
- Package installation;
- Basic concepts: entities, resources, queues, etc.;
- Advanced concepts: resource priority, resource sharing, queue control, resource *Store*, etc.;

- Experimentation (replications, warm-up time, confidence intervals, etc.);
- Agent simulation and optimization;
- AI applications.

From time to time, if it doesn't rain and isn't too sunny, the author commits to updating to new versions of SimPy and Python.

Installing SimPy

Our journey begins with a brief installation tutorial for some programs and libraries useful for SimPy. We have selected the following packages to start the tutorial:

1. Python 3.4
2. Pip
3. SimPy 3.0.10
4. NumPy

A brief preamble on our choices: at the time this tutorial was written, **Python** was at version 3.5.0, but **Anaconda** (explained below) still provided version 3.4. If you already have an installation with Python 3.5 or later, you can install SimPy without problems, as it is compatible with the most recent versions of Python.

Pip is a library installer and makes the programmer's life much easier.

SimPy 3.0.10 is the most current version at the time this tutorial is written and brings major changes compared to version 2.0.

⚠ There is vast material available on the Internet for SimPy. However, special care must be taken: much of this material refers to version 2, which has critical differences from the current version. This text refers to version 3 onwards.

As for **NumPy**, let's take the opportunity to install it, as it will be very useful in our simulation models. Basically, NumPy adds a data type (*n-dimensional array*) that facilitates coding simulation models, particularly in analyzing output data from the model.

Step 1: Anaconda, the easy way

- If you already have Python and Pip installed on your machine, skip directly to Step 3: "Installing SimPy";
- If you already have Python installed, but not Pip (those with Python +3.4 already have Pip installed), skip to Step 2: "Installing Pip"

If this is your first time with Python+SimPy, suggestion: don't waste time and install the free Anaconda distribution.

Anaconda

Through Anaconda, everything is easier, cleaner, and the process already installs more than 200 packages verified for all sorts of compatibility, so you don't have any work. (Among the installed packages is NumPy which, as explained, will be very useful in developing your models).

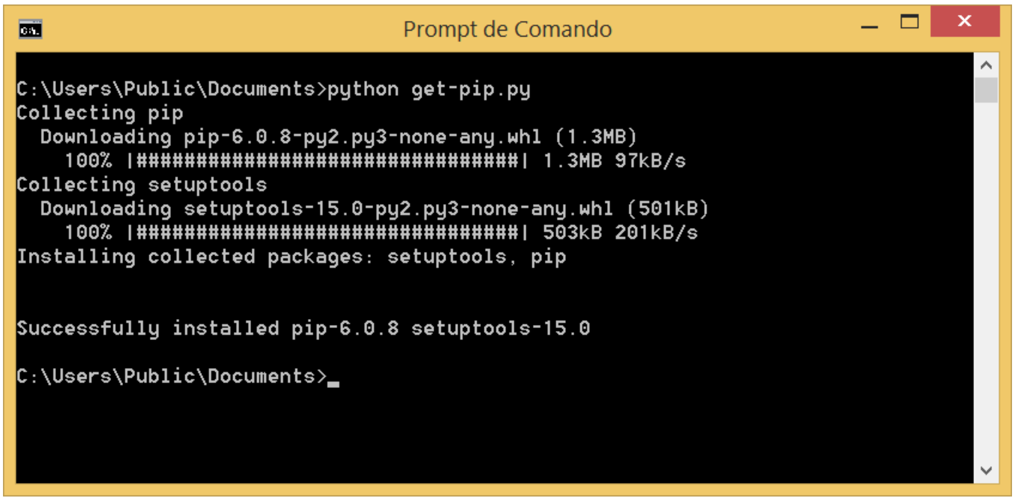
At the time of writing this book, they provided Python versions 2.7, 3.4, and 3.5 (in 32 and 64 bit) on the downloads page.

Download the file with the desired version (once again: SimPy runs on both versions) and follow the installer instructions.

Step 2: Installing Pip (for those who did not install Anaconda)

If the installed Python version is +3.4 or you did the previous step, you can skip this step, as pip has already been installed on your computer.

1. Download the `get-pip.py` package through this link, saving it to a convenient working folder.
2. Run `python get-pip.py` in the chosen working folder (note the message at the end, confirming that pip was successfully installed).



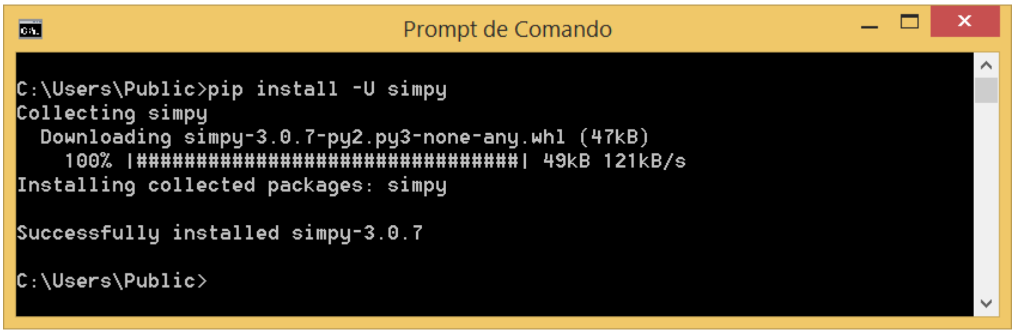
```
C:\Users\Public\Documents>python get-pip.py
Collecting pip
  Downloading pip-6.0.8-py2.py3-none-any.whl (1.3MB)
    100% |################################| 1.3MB 97kB/s
Collecting setuptools
  Downloading setuptools-15.0-py2.py3-none-any.whl (501kB)
    100% |################################| 503kB 201kB/s
Installing collected packages: setuptools, pip


Successfully installed pip-6.0.8 setuptools-15.0

C:\Users\Public\Documents>_
```

Step 3: Installing SimPy

Installing SimPy is easy!

Type in a cmd window:

```
pip install -U simpy
```



```
C:\Users\Public>pip install -U simpy
Collecting simpy
  Downloading simpy-3.0.7-py2.py3-none-any.whl (47kB)
    100% |################################| 49kB 121kB/s
Installing collected packages: simpy

Successfully installed simpy-3.0.7

C:\Users\Public>
```

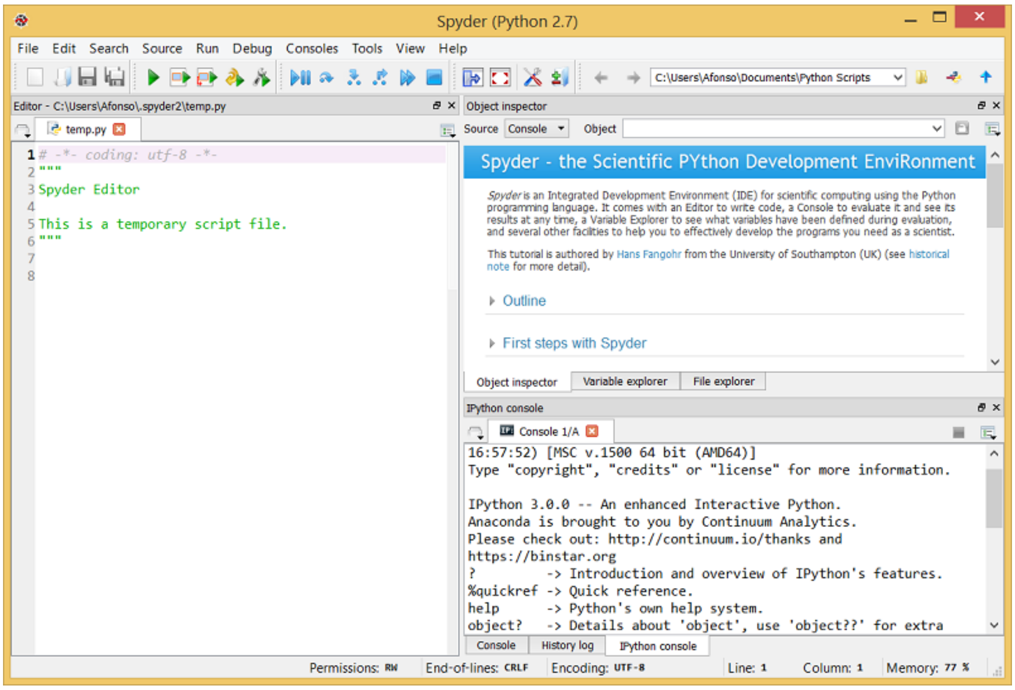
The message “Successfully installed simpy-3.0.10” indicates that you are ready for SimPy. But before that, I have a suggestion for you:

Step 4: Installing an Integrated Development Environment (IDE)

IDEs, for those who don’t know, are true code editing interfaces that make the programmer’s life easier. They usually have an advanced text editor, error checking features, monitors for code variable states, step-by-step processing commands, etc.

If you installed Anaconda, then you already got a good one: Spyder, which is already configured and ready for use. Usually (depending on your Operating System version) it appears as an icon on the desktop. If you can’t find the icon, search for Spyder on your computer (notice the “y” trick) or type the command `spyder` in a **cmd** window.

Once open, Spyder looks like the following figure:


Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Tools View Help
C:\Users\Afonso\Documents\Python Scripts
Editor - C:\Users\Afonso\.spyder2\temp.py
temp.py
Object inspector
Source Console Object
Object inspector Variable explorer File explorer
IPython console
Console 1/A
Console History log IPython console
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 1 Memory: 77 %

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
```

Spyder - the Scientific PYthon Development EnviRonment

Spyder is an Integrated Development Environment (IDE) for scientific computing using the Python programming language. It comes with an Editor to write code, a Console to evaluate it and see its results at any time, a Variable Explorer to see what variables have been defined during evaluation, and several other facilities to help you to effectively develop the programs you need as a scientist.

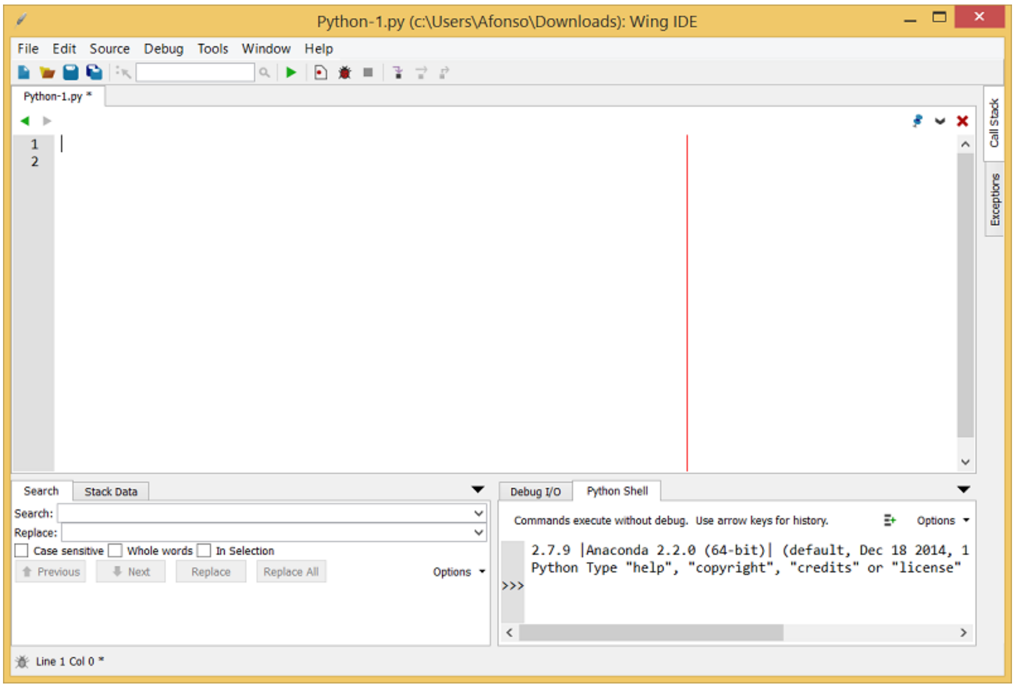
This tutorial is authored by Hans Fangohr from the University of Southampton (UK) (see historical note for more detail).

Outline

First steps with Spyder

```
16:57:52) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 3.0.0 -- An enhanced Interactive Python.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and
https://binstar.org
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
```

Another very good IDE is Wing IDE 101 which is free (and also has a paid professional version):



```
Commands execute without debug.  Use arrow keys for history.
2.7.9 |Anaconda 2.2.0 (64-bit)| (default, Dec 18 2014, 1
Python Type "help", "copyright", "credits" or "license"
>>>
```

If you installed Anaconda and intend to use an IDE other than Spyder, follow the specific configuration instructions at this link: Using IDEs.

If you've made it this far and have everything installed, the next step is to really get started with SimPy!

In the next section, of course. We need a break for a cup of tea after so many installed bytes.

Figure 2. "The Mad Hatter, a character from Alice's Adventures in Wonderland (1865) by John Tenniel"

It All Depends on Python

Before we start with SimPy, we need to make sure you have some minimum knowledge of Python. If you feel your knowledge of the language is reasonable, the recommendation is that you skip to the next section, “Test Your Python Knowledge”.

If you have never had contact with the language, I warn you that I do not intend to build an “introduction” or “tutorial” for Python, simply because that is what is most available on the internet.

Look for a quick tutorial (there are tutorials of up to 10 minutes!) and get to work! Nothing easier than learning the basics of Python.

Suggestions:

1. Sololearn: one of the fastest and coolest ways to learn Python (and other languages). There are several courses, divided by levels, that can even be followed on your phone (yes, I learned Java traveling on the blue line of the SP subway).
2. An Introduction to Interactive Programming in Python (Part 1): Coursera has a course that will teach you to create programs that interact with the user, that is: little games 😂. I took it, it’s good!
3. *Summerfield, Mark. Programming in Python 3: a complete introduction. Addison-Wesley Professional. 2012*: I always have this excellent Python book on my desk. It can be found on Amazon.

After completing the tutorial, course, or even learning everything on your own, test your knowledge to verify if you know the *basic necessary* Python to start with SimPy.

Test Your Python Knowledge: the gambler’s ruin problem

The gambler’s ruin problem is a classic problem proposed by Pascal in a letter to Fermat in 1656. The version presented here is a simplification, aimed at evaluating your Python knowledge.

Challenges

Challenge 1: two gamblers start a heads or tails game in which each of them always bets $1 on the same side of the coin. The winner takes the total bet ($2). Each player initially has $5 available to bet. The game ends when one of the players reaches ruin and has no more money to bet.

Build three functions:

1. `transfer(winner, looser, bankroll, tossCount):` transfers the value from the losing player to the winner and prints the winner's name on the screen;
2. `coinToss(bankroll, tossCount):` draws the winner of heads or tails;
3. `run2Ruin(bankroll):` maintains a permanent loop until one of the players reaches ruin

Test the program with the suggested parameters (you can use the following code as a template for your program):

```python
import random                       # random number generator

names = ['Chewbacca', 'R2D2']       # players

def transfer(winner, looser, bankroll, tossCount):
    # function that transfers money from winner to looser
    # prints the toss winner and each player's bankroll
    pass

def coinToss(bankroll, tossCount):
    # function that tosses the coin and calls transfer
    pass

def run2Ruin(bankroll):
    # function that runs the game until one player's ruin
    pass

bankroll = [5, 5]                   # money available for each player
run2Ruin(bankroll)                  # start the game
```

Now it's up to you: complete the previous code and find out if you are ready to start with SimPy! (The next section presents a possible answer to the challenge and, following that, everything finally begins.)

Solution to Challenge 1

> **Challenge 1:** two gamblers start a heads or tails game in which each of them always bets $1 on the same side of the coin. The winner takes the total bet ($2). Each player initially has $5 available to bet. The game ends when one of the players reaches ruin and has no more money to bet.

The following code is a possible solution to challenge 1 from the previous section. Naturally, it is possible to make it clearer, more efficient, obscure, evil, elegant, faster or slower, like any programming code. The important thing is that if you did something that works, I believe it is enough to start with SimPy.

```python
import random                   # random number generator

names = ['Chewbacca', 'R2D2']   # players

def transfer(winner, looser, bankroll, tossCount):
    # function that transfers money from winner to looser
    # prints the toss winner and each player's bankroll
    bankroll[winner] += 1
    bankroll[looser] -= 1
    print("\nToss: %d\tWinner: %s" % (tossCount, names[winner]))
    print("%s has: $%d and %s has: $%d"
            % (names[0], bankroll[0], names[1], bankroll[1]))

def coinToss(bankroll, tossCount):
    # function that tosses the coin and calls transfer
    if random.uniform(0, 1) < 0.5:
        transfer(1, 0, bankroll, tossCount)
    else:
        transfer(0, 1, bankroll, tossCount)

def run2Ruin(bankroll):
    # function that runs the game until one player's ruin
    tossCount = 0               # toss counter
    while bankroll[0] > 0 and bankroll[1] > 0:
        tossCount += 1
        coinToss(bankroll,tossCount)
    winner = bankroll[1] > bankroll[0]
    print("\n%s won after %d tosses, game over!"
            % (names[winner], tossCount))

bankroll = [5, 5]               # money available for each player
run2Ruin(bankroll)              # start the game
```

On my computer, the previous problem provides the following result:

```
Toss: 1   Winner: Chewbacca
Chewbacca has: $6 and R2D2 has: $4

Toss: 2   Winner: Chewbacca
Chewbacca has: $7 and R2D2 has: $3

Toss: 3   Winner: Chewbacca
Chewbacca has: $8 and R2D2 has: $2

Toss: 4   Winner: Chewbacca
Chewbacca has: $9 and R2D2 has: $1

Toss: 5   Winner: Chewbacca
Chewbacca has: $10 and R2D2 has: $0

Chewbacca won after 5 tosses, game over!
```

Note that the result provided should be different on your computer, just as it changes with each new run of the program. For the results between my computer and yours to be similar, we need to ensure that the sequence of random numbers on both computers is the same. This is possible with the `random.seed(seed)` command, which establishes a fixed initial value for the seed of the generated random number sequence (see item 1 in the "Test Your Knowledge" section below).

Test Your Knowledge

1. Each time you run the program, the `random.uniform(0,1)` function draws a new random number between 0 and 1, making the program result unpredictable. Use the `random.seed(seed)` function to make the generated sequence of random numbers always the same.
2. Add a loop to the main program so that the game can be repeated up to a predefined number of times. Simulate 100 games and check how many games each player won.
3. The Gambler's Ruin Problem is commonly applied in situations of gamblers playing against Casinos. In this situation, the gambler has a finite amount of money and plays against another gambler, the Casino, with a very large or infinite amount. Consider that R2D2 is a Casino and therefore has an infinite *bankroll*, while Chewbacca's is limited to $5. How

would you use the built simulator to demonstrate that the probability of ruin for gambler Chewbacca is equal to 1? (Note that the statement is true even considering that the game is fair and there is no fee charged by the Casino).

First Steps in SimPy: creating entities

Something elementary in any simulation package is a function to create entities within the model. It is the “Hello World!” of simulation packages. Your first mission, should you choose to accept it, will be to build a function that generates entities with exponentially distributed inter-arrival times, with a mean of 2 min. Simulate the system for 10 minutes only.

Importing the random and simpy libraries

Initially, two Python libraries will be needed: `random` – the random number generation library – and `simpy`, which is SimPy itself.

Our first simulation model in SimPy begins with importing the respective libraries of interest:

```python
import random                   # random number generator
import simpy                    # simulation library

random.seed(1000)               # random number generator seed
```

Note the final line `random.seed(1000)`. It ensures that the random number generation will always start from the same seed, so that the sequence of random numbers generated in each program execution will always be the same, facilitating the program verification process.

Creating a simulation environment

Everything in SimPy revolves around **events** generated by functions and all events must occur in an **environment**, or a simulation “environment” created from the `simpy.Environment()` function.

Thus, our program must contain at least one call to the `simpy.Environment()` function, creating an *environment* “env”:

```python
import random              # random number generator
import simpy               # simulation library

random.seed(1000)          # random number generator seed
env = simpy.Environment() # creates the model environment in the variable env
```

If you run the previous program, nothing happens. At the moment, you have only created an *environment*, but you have not created any process, therefore, there is no event yet to be simulated by SimPy.

Creating an arrival generator within the environment

Let's write a function `generateArrivals()` that creates entities in the system for the duration of the simulation. Our first entity generator will have three input parameters: the *environment*, an attribute that will represent the entity name, and the desired rate of entity arrivals per unit of time. For SimPy, this is equivalent to saying that you will build an *event generator function* within the created *environment*. In this case, the generated events will be the arrivals of entities in the system.

Thus, our code begins to take shape:

```python
import random              # random number generator
import simpy               # simulation library

def generateArrivals(env, name, rate):
    # function that creates entity arrivals in the system
    pass

random.seed(1000)          # random number generator seed
env = simpy.Environment() # creates the model environment
```

We need to inform SimPy that the `generateArrivals()` function is, in fact, a process that should be executed throughout the entire simulation. A process is created within the `environment` by the command:

```python
env.process(function_that_generates_the_process)
```

The process call is always made after creating the `env`, so just add a new line to our code:

```python
import random               # random number generator
import simpy                # simulation library

def generateArrivals(env, name, rate):
# function that creates entity arrivals in the system
    pass

random.seed(1000)           # random number generator seed
env = simpy.Environment() # creates the model environment
# creates the arrival process
env.process(generateArrivals(env, "Customer", 2)))
```

Creating time intervals with env.timeout(wait_time)

Initially, we need to generate random time intervals, exponentially distributed, to represent the times between successive entity arrivals. To generate arrivals with exponential intervals, we will use the `random` library, well detailed in its documentation, which has the function:

```python
random.expovariate(lambd)
```

Where `lambd` is the rate of event occurrence or, mathematically, the inverse of the mean time between successive events. In this case, if we want arrivals to occur at mean intervals of 2 min, the function would be:

```python
random.expovariate(lambd=1.0/2.0)
```

The previous line is basically our exponentially distributed random number generator. The next step will be to inform SimPy that we want our entities appearing in the system according to the defined distribution. This is done by calling the reserved word `yield` with the SimPy function `env.timeout(interval)`, which is nothing more than a function that causes a time delay, a *delay* in time within the created *environment* `env`:

```python
yield env.timeout(random.expovariate(1.0/2.0))
```

In the previous line of code we are executing `yield env.timeout(0.5)` so that the model delays the process by a random time generated by the `random.expovariate(0.5)` function.

In due time, we will discuss more deeply the role of the word `yield` (*spoiler*: it is not from SimPy, but originally from Python itself). For now, consider that it is just a way to **create events** within the `env` and that, if a function represents a process, it must necessarily contain the command `yield *something*`, as well as the respective `environment` of the process.

A function created in Python (with the `def` command) is only treated as a **process** or **event generator** for SimPy if it contains at least one line of code with the `yield` command. Later, the section "What are generator functions" explains in more detail how `yield` works.

Putting everything together in the `generateArrivals()` function, we have:

```python
import random                # random number generator
import simpy                 # simulation library

def generateArrivals(env, name, rate):
    # function that creates entity arrivals in the system
    arrivalCount = 0
    while True:
        yield env.timeout(random.expovariate(1.0/rate))
        arrivalCount += 1
        print("%s %i arrives at: %.1f " % (name, arrivalCount, env.now))

random.seed(1000)            # random number generator seed
env = simpy.Environment() # creates the model environment

# creates the arrival process
env.process(generateArrivals(env, "Customer", 2)))
```

The code should be self-explanatory: the `while` loop is **infinite** for the duration of the simulation; a counter, `arrivalCount`, stores the total number of generated entities and the `print` function prints the arrival time of each customer on the screen. Note that, within the `print`, there is a call to the **current simulation time** `env.now`. Finally, a call to the `random.seed()` function ensures that the random numbers in each program execution will be the same.

Running the model for a determined time with env.run(until=simulation_time)

If you run the previous code, nothing happens again, because we still need to inform SimPy what the simulation duration time is. This is done by the command:

```
env.run(until=simulation_time)
```

In the proposed example, the simulation time should be 10 min, as represented in line 15 of the following code:

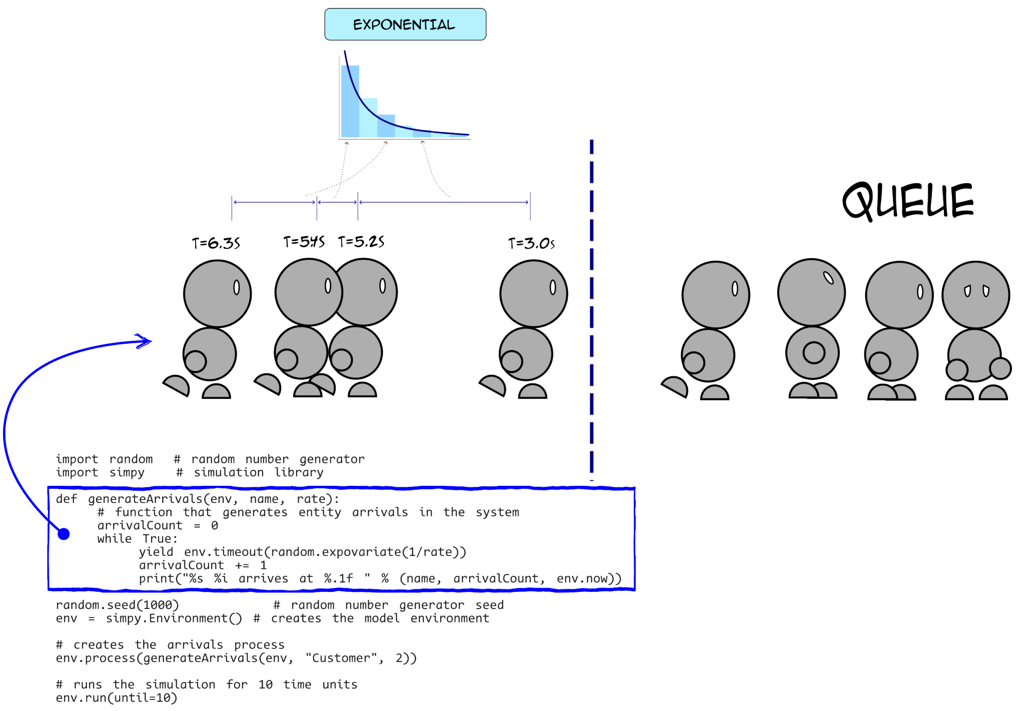
```
 1  import random               # random number generator
 2  import simpy                # simulation library
 3
 4  def generateArrivals(env, name, rate):
 5      # function that creates entity arrivals in the system
 6      arrivalCount = 0
 7      while True:
 8          yield env.timeout(random.expovariate(1/rate))
 9          arrivalCount += 1
10          print("%s %i arrives at: %.1f " % (name, arrivalCount, env.now))
11
12  random.seed(1000)           # random number generator seed
13  env = simpy.Environment() # creates the model environment
14
15  # creates the arrival process
16  env.process(generateArrivals(env, "Customer", 2))
17
18  # runs the simulation for 10 time units
19  env.run(until=10)
```

When running the program, we get the output:

```
Customer 1 arrives at: 3.0
Customer 2 arrives at: 5.2
Customer 3 arrives at: 5.4
Customer 4 arrives at: 6.3
Customer 5 arrives at: 7.6
Customer 6 arrives at: 9.1
```

Now we're talking!

Note that `env.process(generateArrivals(env))` is a command that **turns** the `generateArrivals()` function into a **process** or an **event generator** within the `Environment env`. This process only starts being executed in the next line, when `env.run(until=10)` informs SimPy that every process belonging to `env` should be executed for a **simulation time** equal to 10 minutes.



```
import random   # random number generator
import simpy    # simulation library

def generateArrivals(env, name, rate):
    # function that generates entity arrivals in the system
    arrivalCount = 0
    while True:
        yield env.timeout(random.expovariate(1/rate))
        arrivalCount += 1
        print("%s %i arrives at %.1f " % (name, arrivalCount, env.now))

random.seed(1000)             # random number generator seed
env = simpy.Environment() # creates the model environment

# creates the arrivals process
env.process(generateArrivals(env, "Customer", 2))

# runs the simulation for 10 time units
env.run(until=10)
```

Figure 3. Representation of arrivals in the queue.

Concepts from this section

Content	Description
`env = simpy.Environment()`	creates a simulation `Environment`
`random.expovariate(lambd)`	generates exponentially distributed random numbers, with occurrence rate (events/time unit) equal to `lambd`
`yield env.timeout(time)`	generates a delay given by `time`
`random.seed(seed)`	sets the random seed generator to the same value for each new simulation
`env.process(generateArrivals(env))`	starts the `generateArrivals` function as a *process* in `env`
`env.run(until=simTime)`	executes the simulation (executes all processes created in `env`) for time `simTime`
`env.now`	returns the current instant of the simulation

Challenges

Challenge 2: it is common for entity creation commands in proprietary software to have the option to limit the maximum number of entities generated during the simulation. Modify the `generateArrivals` function so that it receives `maxArrivals` as a parameter and limits the creation of entities to this number.

Challenge 3: modify the `generateArrivals` function so that the inter-arrival times are distributed according to a triangular distribution with mode 1, minimum value 0.1 and maximum value 1.1.

Solution to Challenges 2 and 3

> **Challenge 2:** it is common for entity creation commands in proprietary software to have the option to limit the maximum number of entities generated during the simulation. Modify the `generateArrivals` function so that it receives `maxArrivals` as a parameter and limits the creation of entities to this number.

In this case, the Python *script* is self-explanatory, just note that I limited the number of arrivals to 5 and did this before calling the process generated by the `generateArrivals()` function:

```python
import random      # random number generator
import simpy       # simulation library

def generateArrivals(env, name, rate, maxArrivals):
    # function that creates entity arrivals in the system
    arrivalCount = 0
    while (arrivalCount < maxArrivals):
        yield env.timeout(random.expovariate(1/rate))
        arrivalCount += 1
        print("%s %i arrives at: %.1f " % (name, arrivalCount, env.now))

random.seed(1000)   # random number generator seed
env = simpy.Environment() # creates the model environment
# creates the arrival process
env.process(generateArrivals(env, "Customer", 2, 5))
env.run(until=10) # runs the simulation for 10 time units
```

> **Challenge 3:** modify the `generateArrivals` function so that the inter-arrival times are distributed according to a triangular distribution with mode 1, minimum value 0.1 and maximum value 1.1.

In this case, we need to check the `random` library documentation to see what our options are. The following table summarizes the available distributions:

Function	**Distribution**
`random.random()`	generates random numbers in the interval [0.0, 1.0)
`random.uniform(a, b)`	uniform in the interval [a, b]
`random.triangular(low, high, mode)`	triangular with minimum value *low*, maximum value *high* and mode *mode*
`random.betavariate(alpha, beta)`	beta with parameters *alpha* and *beta*
`random.expovariate(lambd)`	exponential with mean 1/*lambd*
`random.gammavariate(alpha, beta)`	gamma with parameters *alpha* and *beta*
`random.gauss(mu, sigma)`	normal with mean *mu* and standard deviation *sigma*
`random.lognormvariate(mu, sigma)`	lognormal with mean *mu* and standard deviation *sigma*
`random.normalvariate(mu, sigma)`	equivalent to random.gauss, but slightly slower
`random.vonmisesvariate(mu, kappa)`	von Mises distribution with parameters *mu* and *kappa*
`random.paretovariate(alpha)`	pareto with parameter *alpha*
`random.weibullvariate(alpha, beta)`	weibull with parameters *alpha* and *beta*

The NumPy library, which we will see in due time, has more options for statistical distributions. For now, challenge 3 can be solved literally:

```python
import random     # random number generator
import simpy      # simulation library

def generateArrivals(env, name, maxArrivals):
    # function that creates entity arrivals in the system
    arrivalCount = 0
    while (arrivalCount < maxArrivals):
        yield env.timeout(random.triangular(0.1,1,1.1))
        arrivalCount += 1
        print("%s %i arrives at: %.1f " % (name, arrivalCount, env.now))

random.seed(1000)          # random number generator seed
env = simpy.Environment() # creates the model environment
# creates the arrival process
```

```python
env.process(generateArrivals(env, "Customer", 5))
env.run(until=10)
```

Tip: simulation models, with many arrival and service processes, tend to use several different probability distribution functions, making things a bit confusing for the programmer. A cool tip is to create a function that stores all the model's distributions in a single place, like a distribution shelf.

For example, imagine a SimPy model that has 3 processes: one exponential with mean 10 min, one triangular with parameters (10, 20, 30) min, and one normal with mean 0 and standard deviation 1 minute. The following `distribution()` function stores all random number generators in a single location:

```python
import random

def distributions(dtype):
    return {
        'arrival': random.expovariate(1/10.0),
        'singing': random.triangular(10, 20, 30),
        'applause': random.gauss(10, 1),
    }.get(dtype, 0.0)
```

The next example tests how to call the function:

```python
import random

def distributions(dtype):
    return {
        'arrival': random.expovariate(1/10.0),
        'singing': random.triangular(10, 20, 30),
        'applause': random.gauss(10, 1),
    }.get(dtype, 0.0)

dtype = 'arrival'
print(dtype, distributions(dtype))

dtype = 'singing'
print(dtype, distributions(dtype))

dtype = 'applause'
print(dtype, distributions(dtype))
```

Which produces the output:

```
arrival 6.231712146858156
singing 22.192356552471104
applause 10.411795571842426
```

That was our tip of the day!

Feel free to implement random number generation functions to your liking. Note, and this is important, that **practically all your SimPy simulation models will need this type of function!**

Test Your Knowledge

1. Add to the initial program a `distribution` function as proposed in the “Tip” section and make the time between successive entity arrivals call the function to get the correct value.
2. Consider that 50% of the entities generated during the simulation are female and 50% are male. Modify the program so that it randomly assigns the gender of customers. Do this assignment inside the `distribution` function already created.

Creating, occupying and releasing resources

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

Creating

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

Occupying

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

Releasing

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

Resource status

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

Concepts from this section

Putting it all together in an example: the M/M/1 queue

Entity arrival generation

Performing service at the server

An alternative representation for occupying and releasing resources

Concepts from this section

Challenges

Solution to Challenges 4, 5 and 6

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

Test Your Knowledge

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

Attributes and variables: differences in SimPy

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

Attributes in object-oriented models

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

Concepts from this section

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

Challenges

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

Solution to Challenges 7 and 8

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

Test Your Knowledge

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

Environments: controlling the simulation

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

Execution control with env.run(until=end_of_simulation)

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

Stop by execution of all scheduled processes

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

Stop by end of specific process execution with env.run(until=process)

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

Step-by-step simulation: peek & step

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

Concepts from this section

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

Challenges

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

Solution to Challenges 9 and 10

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

Other types of resources: with priority and preemptive

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

Resources with priority: PriorityResource

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

Resources that can be interrupted: PreemptiveResource

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

Concepts from this section

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

Challenges

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

Solution to Challenges 11 and 12

Process interruptions: simpy.Interrupt

Creating equipment breakdowns

Interrupting a process without capture by try…except

Concepts from this section

Challenges

Solution to Challenges 13 and 14

Test Your Knowledge

Storage and selection of specific objects with Store, FilterStore and PriorityStore

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

Building a set of objects with Store

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

Selecting a specific object with FilterStore()

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

Creating a Store with priority: PriorityStore

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

Concepts from this section

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

Challenges

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

Solution to Challenges 15 and 16

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

Test Your Knowledge

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

Filling or emptying boxes, tanks, stocks or objects with Container()

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

Filling my container yield myContainer.put(quantity)

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

Emptying my container: yield myContainer.get(quantity)

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

Creating a sensor for the current container level

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

Concepts from this section

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

Challenges

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

Solution to Challenges 17 and 18

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

Test Your Knowledge

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

Creating batches (or grouping) entities during simulation

A tactic for batch grouping using Container

Grouping batches by entity attribute using FilterStore

Challenges

Solution to Challenges 19 and 20

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

Test Your Knowledge

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

Creating, manipulating and triggering events with event()

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

Creating an isolated event with event()

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

Concepts from this section

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

Challenges

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

Solution to Challenges 21 and 22

Test Your Knowledge

Waiting for multiple events at the same time with AnyOf and AllOf

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

Waiting until at least one event finishes with AnyOf

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

Waiting for all events with AllOf

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

Better understanding the outputs of AllOf and AnyOf commands

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

Concepts from this section

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

Challenges

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

Solution to Challenges 23 and 24

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

Test Your Knowledge

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

Useful properties of events

Concepts from this section

Challenges

Solution to Challenge 25

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

Test Your Knowledge

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

Adding callbacks to events

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

Every process is an event

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

Concepts from this section

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

Challenges

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

Solution to Challenge 26

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

Test Your Knowledge

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

Event interruptions

Interrupting an event with the interrupt method

Interruption control method 1: try…except exception logic

Interruption control method 2: changing the defused attribute

Interrupting an event with the fail method

What are generator functions? (or how SimPy works) - Part I

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

Iterator

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

Generator functions

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

What are generator functions? (or how SimPy works?) - Part II

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

SimPy vs. generator functions

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

Agent-Based Simulation using SimPy

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

How to build an agent simulation model: basic steps

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

Agent-based epidemic model: the SIR model[1]

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

Modeling the problem in SimPy

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

Library import

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

Input parameters and global variables

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

[1]Based on: Borshchev, A. and Grigoryev, Ilya. The Big Book of Simulation Modeling. Multi-method modeling with AnyLogic 8. AnyLogic North America. 2020.

Agent constructor

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

Initialization

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

Sending and receiving messages

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

Infection process

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

Incubation process

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

Monitoring and displaying graphs

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

Execution (phew!)

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

Improving simulation code performance

Creating a mesh of connections between nearest neighbors

How to proceed from here

Concepts from this section

Challenges

Solution to Challenges 27 and 28

Test Your Knowledge

Data input and output via spreadsheet

Communication with the library