# Writing Better Python3

M3AAWG 61 Vienna Austria 90 minutes

Joe St Sauver, Ph.D. M3AAWG Expert Advisor



https://xkcd.com/353/ [license: https://creativecommons.org/licenses/by-nc/2.5/]

### **On Writing Python3 Code**

- Virtually anyone can learn to write Python3 code. Python3's a very forgiving language with excellent documentation and exceptionally clear and informative error messages.
- Learning to code Python3 is like learning to fly in a good trainer aircraft: it seems designed to self-correct and works hard to keep you "in the air" and under control.
- We don't have the time to make you into a full-fledged professional Python3 coder, but we'll review some basics and hopefully share at least a few new "tips" and "tricks" you can add to your existing Python3 skills (if any).
- We're generally going to assume that you're working on a Mac laptop, but most of what we discuss will be equally applicable to those of you using Windows Subsystem for Linux under MS Windows (https://learn.microsoft.com/en-us/windows/wsl/install) or to those of you using a Linux distribution such as Debian or RedHat.

## There Are a Lot of Things We're NOT Going To Cover Today

- We're going to cover a lot of material today, but we're not going to have time to cover...
  - Using databases in Python3 (neo4j, PostgreSQL, Reddis, etc.)
  - Doing graphics with Seaborn, Matplotlib or other Python3 graphic APIs
  - Writing GUI Python3 applications with tkinter, PyQT5, etc.
  - Integrating Python3 with some other language, such as C
  - Machine learning in Python3 using Keras, PyTorch, Scikit-learn, or tensor-flow
  - Making maps with cartopy or some other Python3 mapping package
  - Message passing applications using MPI in Python3
  - Python3 in the "cloud" (e.g., we're not covering boto3, google-API-core, etc.)
  - Scientific computing with Pandas, Numpy, SciPy, etc.
  - Streaming applications in Python3 (using Kafka, pyzmq, etc.)
  - Writing threaded/multiprocessing parallel Python3 code
- Today's training will hopefully get you ready to dig into some of these others areas later!

#### **Copying & Pasting Code Snippets From Presentations**

- If you try cutting and pasting code from PowerPoints or PDFs, it will usually work fine.
- However, sometimes "what you see" may not be "what you end up getting" due to some programs attempting to produce "pretty" documents using ligatures ("combined letters") or other font substitutions (such as "smart quotes" for "straight quotes").
- Newlines may also end up replaced with special characters.
- We've endeavored to disable this behavior to the maximum extent we can, but some examples of this behavior may have slipped through.
- If you run into that, sorry about that. You may need to clean those up manually.

## **Today's Itinerary**

1.	Untangling Your Python3 Installation	6
2.	<b>Options for Writing Python3 Programs</b>	13
3.	Python3 and LLMs	18
4.	Statements	28
5.	I/O	43
6.	Simple Data Types	56
7.	Lists	60
8.	Dictionaries	67
9.	Other Python3 Data Structures	73
10.	Packages/Libraries	78
11.	Argparse	90
12.	Sanitizing/Validating/Encoding Inputs/Fuzzing	94
13.	Other Python3 Testing	100
14.	Exercises	106

#### 1. Untangling Your Python3 Installation

Do you already have Python3 installed? What version(s)? Should you upgrade?

Which version are you running by default? How can you change the default version?

What if you *must* use some newly-released feature?



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

https://xkcd.com/1987/ [license: https://creativecommons.org/licenses/by-nc/2.5/]

### Python3 on <u>Your</u> System

- *Caution:* Depending on the system you're using, your system may come with a copy of Python3 already pre-installed. That pre-installed Python3 will often be used for system utilities and maintenance tools. Even though that pre-installed Python3 may be an older version, don't mess around with it. Let the system itself handle updating and managing any bundled version of Python3.
- To check the version of Python you'll run by default, if any, open a terminal window (e.g., on a Mac, Applications → Utilities → Terminal) and then enter:



• You'll want to know the version of Python3 you've got installed because some functions and options may only be supported for certain versions of Python3.

• You may even have **multiple** versions of Python3 pre-installed on your system. To check:

```
$ whereis -a -b python3
```

python3: /usr/bin/python3

/Library/Frameworks/Python.framework/Versions/3.11/bin/python3
/Library/Frameworks/Python.framework/Versions/3.12/bin/python3
/opt/homebrew/bin/python3 /usr/local/bin/python3

• Those copies of Python3 may represent **different versions**. For example:

```
$ /opt/homebrew/bin/python3 -V
Python 3.12.2
$ /usr/local/bin/python3 -V
Python 3.11.8
$ /usr/bin/python3 -V
```

Python 3.9.6

• Other "copies" may actually just be "convenience" links:

```
$ ls -l /usr/local/bin/python3
lrwxr-xr-x 1 root wheel 70 Mar 13 16:49 /usr/local/bin/python3 ->
../../Library/Frameworks/Python.framework/Versions/3.11/bin/python3
```

# Which Specific Installation of Python3 Will I Run By Default?

• We talked previously about what **VERSION** of Python3 you'd run by default, but **WHERE** is that program located on your system? To see the LOCATION of the Python3 you'll run by default (if you don't provide a full path to something else), use the which command:

#### \$ which python3

/Library/Frameworks/Python.framework/Versions/3.11/bin/python3

- If you want to change the default version of Python3 you run, change the order of the directories listed in your system's PATH in .bash\_profile (or whatever's the appropriate hidden "dot configuration" file for the shell you've chosen to run).
- Invoke a specific version of Python for just **one** run, assuming that version's installed and available on your system by mentioning that specific version when invoking Python3:

\$ python3.11

(instead of just saying python3)

### What About Upgrades? How To Install NEW Versions of Python3

- Most users should run a currently supported production version of Python3 (such as Python 3.11 or 3.12). One good article on this: "Have you updated your Python recently?" https://www.mostlypython.com/have-you-updated-your-python-recently/
- If your device is self-managed (and not centrally managed by your IT team on your behalf), you should be able to upgrade or install new versions of Python3 using either a standalone installer, or using a package manager (such as Homebrew, assuming you're on a Mac).

Either approach will work, but we generally encourage folks to download official standalone installers from the Python web site:

https://www.python.org/downloads/

• Building Python3 from source is possible but complex. Avoid attempting to do this.

## Feature Awareness and Explicitly Handling Required Versions

 Python routinely adds new features, and periodically deprecates or removes old features (see https://docs.python.org/3/whatsnew/index.html). The module index and the documentation for individual routines will call out deprecated module status information:

https://docs.python.org/3/py-modindex.html			
	m		
	mailbox	Manipulate mailboxes in various formats	
	mailcap	Deprecated: Mailcap file handling.	

```
mailcap — Mailcap file handling ¶
Source code: Lib/mailcap.py
Deprecated since version 3.11, will be removed in version 3.13: The mailcap module is deprecated (see PEP 594 for details). The mimetypes module provides an alternative.
Mailcap files are used to configure how MIME-aware applications such as mail readers and web browsers react to files with different MIME types. (The name "mailcap" is derived from the phrase "mail capability".) For example, a mailcap file might contain a line like video/mpeg; xmpeg %s. Then, if the user approximates an amail message or web document with the MIME type video/mpeg, %s will be re-
```

• If you **ABSOLUTELY MUST** use version-specific features (such as new and/or about-to-be-deprecated features), build explicit run-time version checks into your code, such as:

```
import sys
if sys.version_info < (3,11):
    # pylint: disable-next=broad-exception-raised
    raise Exception("Must use Python 3.11 or later")</pre>
```

#### "What About Anaconda, etc.?"

 Anaconda is a popular option for installing and working with Python3 and frameworks such as NumPy and SciPy, however Anaconda limits free use in commercial environments (requiring a paid license for most commercial use). See

https://legal.anaconda.com/policies/en/?name=anacondacontracting-hub#purchased-vs-free-offerings

• YOU may potentially qualify for free Anaconda access (e.g., for non-commercial use, for personal use, for educational use, etc), but due to the commercial affiliation of most attendees and M3AAWG's stict non-commercialization policies, we will not be considering Anaconda and related frameworks today.

#### 2. Options for Writing Python3 Programs

Using IDEs, Notebooks, Editors Coding Style and PEP 8 Linting



https://xkcd.com/378/

[license: https://creativecommons.org/licenses/by-nc/2.5/]

## Picking A Way To Write Python3

- IDEs:
  - You can use an Integrated Development Environments ("IDE"): Many IDE's are paid commercial products (not discussed due to M3AAWG's non-commercial policies). Beware: the learning curve for IDE's may sometimes (initially) be pretty steep.
  - One free Python3 IDE that's available is PyCharm CE ("Community Edition"), see https://www.jetbrains.com/pycharm/download/ (scroll to the bottom).
  - Python3 also bundles a free (very basic) IDE called IDLE. If you have Python3 installed, you should be able to just say \$ idle3 in a terminal window to try it.
- Interactive "notebooks:"
  - Other Python3 users may swear by an interactive "notebook" style inteface such as the free one provided via Jupyter Notebooks. See: https://jupyter-notebook.readthedocs.io/en/stable/
- You can also just use standard text editor such as vi, emacs, etc. This is what I usually do.
  - You can also try "bolting on" a variety of text editor "add-ons" to "help you" write code, but some of these are rather "gimmicky" and none are essential.

### **Follow PEP 8 Coding Style**

- Normal Python3 coding style varies in many ways from some well-known coding conventions (such as Linux Kernel coding style). Adherence to "normal Python3 coding style" is key to keeping your Python3 code readable and maintainable.
- The "catechism" for Python3 is PEP 8, "Style Guide for Python Code." It's available at

https://peps.python.org/pep-0008/

it will teach you a LOT about how to write well-formatted Python3 (and how to **avoid** writing **badly formatted** Python3).

- Use a Python3 linting program (such as pylint, see https://pypi.org/project/pylint/) to systematically flag any departures from PEP 8.
- Be aware that some Python linting program scoring may be somewhat "twitchy..."

# helloworld.py: one line of Python3 -> a 0/10 score from pylint...

• Every developer in the world has created a "Hello, world!" program when learning to use a new programming language. For example, in Python3:

```
$ cat helloworld.py
print("Hello, world!")
$ python3 helloworld.py
Hello, world!
```

• But if you pylint that code, you'll get a 0/10 rating!

```
$ pylint helloworld.py
********* Module helloworld
helloworld.py:1:0: C0114: Missing module docstring (missing-module-
docstring)
```

```
Your code has been rated at 0.00/10 (previous run: 0.00/10, +0.00)
```

### helloworld.py: add a doc string to then get a 10/10 from pylint

The "cataclysmic problem" here? A missing doc string:

```
$ cat helloworld.py
""" Everyone's favorite first program """
print("Hello, world!")
```

#### \$ pylint helloworld.py

Your code has been rated at 10.00/10 (previous run: 0.00/10, +10.00)

Other program checkers may want still further stylistic tweaks. For example, black (https://github.com/psf/black), "The Uncompromising Code Formatter" insists that there should be a blank line between the doc string and the print command, too. Some may even lecture you about use of single vs double quote marks for strings.

#### 3. Python3 and LLMs

Risks of Using An LLM to Write Code

Dealing with **Mechanical Issues** When Cutting-and-Pasting LLM Generated Code

LLM-Generated Code May Have Stylistic Problems

LLMs May Make Wrong Assumptions

LLMs Solutions May be Initially Be "Minimally Responsive" (Unless Pushed) THE **SIMPLE ANSWERS** TO THE QUESTIONS THAT GET ASKED ABOUT EVERY NEW TECHNOLOGY:

WILL MAKE US ALL GENIUSES?	NO
WILL MAKE US ALL MORONS?	NO
WILL DESTROY WHOLE INDUSTRIES?	YES
WILL MAKE US MORE EMPATHETIC?	NO
WILL MAKE US LESS CARING?	NO
WILL TEENS USE FOR SEX?	YES
WERE THEY GOING TO HAVE SEX ANYWAY?	YES
WILL DESTROY MUSIC?	NO
WILL DESTROY ART?	NO
BUT CAN'T WE GO BACK TO A TIME WHEN-	NO
WILL BRING ABOUT WORLD PEACE?	NO
WILL CAUSE WIDESPREAD ALIENATION BY CREATING A WORLD OF EMPTY EXPERIENCES?	WE WERE ALREADY ALIENATED

https://xkcd.com/1289/ [license: https://creativecommons.org/licenses/by-nc/2.5/]

### Using An LLM To Write Code

- **ChatGPT** and other "large language models" can now be used to produce Python3 code. They may at least help you get a **starting point** for some coding projects.
- If you do decide to use a LLM as a **"programming partner,"** carefully review and test the code it provides. Just like working with any coding partner do NOT assume that the code you get from ChatGPT or any LLM will even run! YOU are still responsible for all the code you deliver, even if your LLM "partner" drafted some (or all) of it for you.
- Mechanical copy/paste issue? if you get code from ChatGPT (or a similar LLM) and then try pasting it into a vi or vim text editor window, you may find that it "auto-indents excessively" or "pastes weird." If you run into that, try setting vi "paste" mode first:

:set paste

When done pasting, you can turn paste mode back off (if you want to) with

:set nopaste

#### A Sample LLM Python Code Generation Project (In One Sentence)

- "Write sample Python3 code to read in and parse sample spam email messages."
- We tried the free version of Chat GPT 3.5 that's available at

https://chat.openai.com/

### Sample ChatGPT Output (pylint rated the first code at 5.95/10.0)

#### ChatGPT 3.5 ~

You

Write sample Python3 code to read in and parse sample spam email messages.



Sure, here's a sample Python code to read and parse sample spam email messages using the `email` module:



#### Even After Repeated Iterations, ChatGPT Code STILL Didn't Follow Basic Python3 Coding Standards

```
$ pylint test5.py
********** Module test5
test5.py:43:0: C0303: Trailing whitespace (trailing-whitespace)
test5.py:46:0: C0303: Trailing whitespace (trailing-whitespace)
test5.py:56:0: C0305: Trailing newlines (trailing-newlines)
test5.py:1:0: C0114: Missing module docstring (missing-module-docstring)
test5.py:4:16: W0621: Redefining name 'file path' from outer scope (line 48) (redefined-
outer-name)
test5.py:16:8: W0621: Redefining name 'subject' from outer scope (line 49) (redefined-
outer-name)
test5.py:17:8: W0621: Redefining name 'sender' from outer scope (line 49) (redefined-
outer-name)
test5.py:18:8: W0621: Redefining name 'receiver' from outer scope (line 49) (redefined-
outer-name)
[etc]
          ____
Your code has been rated at 7.30/10 (previous run: 7.30/10, +0.00)
```

#### You can obviously clean those up, but you shouldn't have to handle mechanical issues.

#### More Substantially, It Simply "Assumed" I'd Be Using Maildir Format

ChatGPT 3.5 ~

#### You

Does that solution assume mbox format?



Apologies for any confusion. The provided solution assumes individual email files, not the mbox format. If your spam emails are in mbox format, we would need to adjust the code accordingly to parse emails from the mbox file.

Here's how you can modify the code to parse emails from an mbox file:



#### And What About MIME Multi-part Messages?

#### ChatGPT 3.5 ~

#### J You

I notice some objects still come out shown as Body: [<email.message.Message object at 0×7f7c3af296a0>, <email.message.Message object at 0×7f7c3af29490>]

#### ChatGPT

If you're encountering `email.message.Message` objects being displayed instead of the actual email body, it's likely because the email body is multi-part or encoded in a way that's not directly readable. To handle such cases, you need to extract the payload of the email message.

Let's modify the code to handle multi-part messages and extract the body properly:



### Full headers? (we ARE working with spam samples, after all)

#### ChatGPT 3.5 ~

J You

That doesn't show me the full headers I'd need to work spam incidents!

#### ChatGPT

If you need to access the full headers of the emails, you can directly access the `items()` method of the message object, which returns a list of tuples containing the email headers and their values. Here's how you can modify the code to include the full headers:



#### Human Users -> Should Get Human-formatted Output...

ChatGPT 3.5 ~

#### J You

Those headers aren't formatted very nicely. Its hard for humans to read raw arrays.

#### ChatGPT

You're correct. Let's format the headers nicely to make them more human-readable. We can iterate through the header tuples and print them in a readable format. Here's the modified code:



Headers: Return-Path: <test@nero.be> X-Original-To: Delivered-To: Received: from mail.nero.be (mail.nero.be [194.78.167.160]) by (Postfix) with ESMTP id E4CF52C6E4 for ; Fri, 23 Dec 2022 15:24:59 Message-ID: <1095615776-1905577984@mail.nero.be> X-Footer: bmVyby5iZQ== Received: from [37.187.205.90] ([192.168.122.1]) by mail.nero.be (Kerio Connect 9.3.1) with ESMTP; Sun, 16 Oct 2022 17:17:07 +0200 Content-Type: text/plain; charset="iso-8859-1" MIME-Version: 1.0 Content-Transfer-Encoding: guoted-printable Content-Description: Mail message body Subject: Jackpot To: Recipients <test@nero.be> From: "Mackenzie Scott" <test@nero.be> Date: Sun, 16 Oct 2022 08:17:03 -0700 Reply-To: dbbs@dbbs12.wecom.work Status: RO X-Status: X-Keywords: X-UID: 18

Sample output from the code... It DID (eventually) do what we'd wanted it to do...

Subject: Jackpot Sender: "Mackenzie Scott" <test@nero.be> Receiver: Recipients <test@nero.be> Date: Sun, 16 Oct 2022 08:17:03 -0700 Body: Dear Winner,

I am Mackenzie Scott, ex-wife of Amazon founder and CEO(Jeff Bezos).I'm donating \$ 4 billion to charities, individuals and colleges across the Globe from the Scott's Foundation, to provide immediate support to people suffering economically from the COVID-19 pandemic and you are one of the lucky winners. I have a donation grant worth \$1,850,000 for you and your community.

Contact me for more information if you are interested.

Best regards Mackenzie Scott

#### 4: Quick Review of Basic Python3 Statements: You Don't Need to Know "Everything" To Have a Working Knowledge of Python3

#### DOES FEATURE USAGE MIRROR THE PARETO PRINCIPLE?

To see how feature usage maps to the 80/20 rule, we organized the data into four categories: frequent, moderate, rare, and never used.



Source: https://go.pendo.io/rs/185-LQW-370/imagess/ Pendo\_Whitepaper\_2020\_FeatureAdoptionReport\_Refresh.pdf

#### Statements

- Python3 has a compact grammar of routinely-used statements (plus a lot of additional obscure features that most people never use). The ten key non-I/O statement types are:
  - 1) Assignment (=)
  - 2) Code Blocks (indentation)
  - 3) Comments (#)
  - 4) Error handling with try/except
  - 5) for loops (plus break and continue)
  - 6) if / elif / else (conditional statement execution)
  - 7) import
  - 8) User-Defined Functions (def / return (and invoking those functions))
  - 9) while loops
  - **10)** with

I/O statements will be handled seperately in the next section

# 1) Assignment Statements (=)

 Like most programming languages, Python3 has the ability to assign values to variables (see next slide for rules around variables). Unlike many programming languages, Python3 variables do NOT need to be typed – their type will be "inferred."

```
payment_amount = 1100
payer = "Joe's Garage"
available_colors = ["red", "white", "blue", "green", "black"]
```

• When assigning multiple values, you may be tempted to put multiple statements per line:

a = 1; b = 4; c = 3.4; d = 8.7; e = 12

While Python3 will LET you do that, **DON'T.** One statement per line, please!

• For an overly-complex discussion of what should be a very simple topic, see: https://docs.python.org/3/reference/simple\_stmts.html#assignment-statements

### Variable Names

"There are only two hard things in Computer Science: cache invalidation and **naming things.**" -- Phil Karlton [emphasis added]

- Variables can be virtually any length (but in practice, keep them to a "reasonable" length). If you frequently mis-spell/confuse some variables, they're probably too long or too similar.
- It's not a "crime" if some variables (such as loop indicies) are short: i, j, k, x, y, etc.
- Variables are case sensitive (UPPER and miXeD are different than lower).
- Besides letters, you can also use numbers or underbars (but variables **must start** with a letter, and **should NOT** have leading or trailing underbars)
- Variables cannot be a reserved word, see https://docs.python.org/3/reference/lexical\_analysis.html at 2.3.1

Some reserved words are incredibly common (I bumped into this when building a mapping app – using "or" was "out" for an Oregon variable, ditto "in" for an Indiana variable)

# 2) Blocks of Code

- Some programming languages set off blocks of statements using curly braces or tags such as begin / end. Not Python3! Python3 uses indentation to set off blocks of statements. Indent 4 spaces to start a new block. DON'T use tabs!
- Random/unnecessary indentation simply isn't allowed:



- Continuation lines are either inferred (as a result of a "dangling commas" or other dangling operator), or explicitly called out by ending the line with a backslash (my preference).
- Do NOT have whitespace at the end of lines (Pylint will whine about it).

# 3) Comments

• Anything after a # sign is just a comment (unless the # is inside a quoted string literal)

# https://mathworld.wolfram.com/GelfondsConstant.html
# bonus factoids: https://xkcd.com/217/ and https://xkcd.com/179/
GELFONDS = 23.140692632 # this is e to the pi power
gelfonds\_constant\_popularity\_ranking = "#1" # obviously true

- Don't lie in your comments
  - Lies may be the result of **imprecision** (the decimal value above is an approximation)
  - Other lies may be total fabrications (popularity ranking is obviously wrong)
  - You can also "lie by **distraction**" (the XKCDs are only marginally relevant)
  - Wrong comments may also be the result of code maintenance w/o comment updates

# 4) Error handling with try / except

```
$ cat sqr rt.py
from math import sqrt
my value = input("enter value: ")
my value = float(my value)
try:
    sqr rt = sqrt(my value)
    print(sqr rt)
except:
    print("couldn't compute square root for", my value)
$ python3 sqr rt.py
enter value: -3
couldn't compute square root for -3.0
```

It is preferable to specify the exact condition the except statement is meant to handle. (See a list of exceptions at docs.python.org/3/library/exceptions.html ). If taking that approach, you may need to have multiple except stanzas.

### EAFP vs LBYL: What Kind of Programmer Are <u>YOU?</u>

• There are two schools of thought in programming:

**EAFP** (Easier to Ask Forgiveness than Permission) vs. **LBYL** (Look Before You Leap).

- EAFP people really love try/except. If you attempt to divide by zero, yeah, not gonna work, so we'll deal with that (hopefully uncommon) corner case if it happens to occur. That's why try/except exists, right? Keep your code simple, short, and sweet...
- That EAFP "way of life" really drives the LBYL people nuts. To the LBYL people, you should have checked the potential operand for known bad values before ever attempting that mathematical operation. Sure there's additional overhead to checking for bad values when most values are going to be just fine, but it's a tiny amount of extra work and "good craftsmen" don't routinely rely on "emergency safety nets" to save them from known-bad situations they could have avoided encountering in the first place.

## 5) for loops – there are two main types of for loops in Python3

• Numeric for loop:

```
for i in range(0, 5): # generates integer values from 0 to 4
print(i)
```

• Iterative **for** loop, looping over values in an iterable (such as a list):

my\_list\_of\_zip\_codes = [97405, 97408, 97412, 97413, 97415]
for zip\_code in my\_list\_of\_zip\_codes:
 print(zip\_code)

- for loops may be nested, just remember to further-indent each subsequent for loop (each for loop starts a new block of code).
- The **break** statement lets you immediately and completely **exit** the current for loop.
- The continue statement lets you start the next iteration of the for loop immediately.
## 6) if / elif / else

• Conditional statements are another "programming language staple." Python has 'em, too.

```
if gpa >= 3.9:
    grad = "Summa Cum Laude"
elif gpa >= 3.7:
    grad = "Magna Cum Laude"
elif gpa >= 3.5:
    grad = "Cum Laude"
elif gpa >= 2.0:
    grad = "No honors"
else:
    grad = "Didn't graduate"
```

- If you want to check for an exact value, use == (not =, = is used to assign values)
- NOT equal is represented by != (common alternatives such as <> or ~= won't work)
- "Write it like you say it:" "greater than or equal to" (NOT =>)
- Use and and or (rather than & and |)
- A departure from PEP 8: when in doubt, use parentheses to clarify logical expressions.

# 7) import

- import lets you bring in a predefined library or "package" such as the Python3 math library (we'll talk more about packages in section 10).
- To import an entire library, simply supply the name of the library: import math
- In general, however, you should only import the *particular function(s)* you need:

```
from math import pi, sin
# remember, the trig fn arguments are in radians, not degrees!
print(sin(pi/4))
0.7071067811865475
```

- If an imported function "can't be found", mention the library name when referring to it: print(math.sin(math.pi/4))
- Some libraries may be routinely associated with a shortened name when imported:

```
import matplotlib.pyplot as plt
[...]
plt.show()
```

8) User-Defined Functions in Python

```
$ cat dbl sqr root.py
from math import sqrt
def double sq rt(my value):
    try:
        result = sqrt(sqrt(float(my value)))
        return(result)
    except:
        print("No double square root possible for ", my value)
# try out our new sample function
test value = double sq rt(17.3)
print(test value)
```

\$ python3 dbl\_sqr\_root.py 2.0394427838547187

# Variable Scoping ("LEGB" → local, enclosing, global, built-in)

- Variables created in a user-defined function are local to that function, and unknown outside of that function.
- Values from outer enclosing functions are available within any inner nested function(s).
- Variables created in the main body of a Python3 program are "globally scoped," readable in the main body and in any function. Avoid trying to update the value of global variables in functions unless you explicitly declare the variables as global in that function.
- If you <u>create</u> the SAME variable name in a function and in the main body of a Python3 program, those variables will be treated as being independent variables (unless the variables are declared to be global in the function). Avoid this confusing situation!
- There are 35 built-in **reserved tokens** you may NOT use for variables. They're hard to find on the Python site (https://docs.python.org/3/reference/lexical\_analysis.html sec 2.3.1)
- Most names that begin with \_ are special. You should usually NOT use names that begin with an underbar in most of your programs. See section 2.3.2 of the just-referenced page.
- Want to see defined variables? print(local())(or print(global()), or print(dir()))

## 9) while: repeat a block of code until a condition is met

```
$ cat approx pi.py
""" how many 'darts' do we need to throw at a unit circle to approximate Pi? """
from random import uniform
approx real pi = 3.14159265359
tolerance = 0.00005
diff = 1000000
iterations = 0
inside the circle = 0
test pi = 0
while diff > tolerance:
   # throw a dart
   x = uniform(0, 1)
    y = uniform(0, 1)
    radius = (x^{*}2) + (y^{*}2)
    if radius < 1:
        inside the circle = inside the circle + 1
    iterations = iterations + 1
    test pi = 4*(float(inside the circle)/float(iterations))
    diff = abs(approx real pi - test pi)
print("it took ", iterations, " darts to approximate pi=", test pi)
```

# 10) with

- with is meant to help you avoid dangling open files (or other dangling resources)
- If YOU are conscientious about remembering to close ALL files you open, you can ignore this statement entirely (I analogize this to the automatic "turn off the headlights" feature on most modern vehicles – if you are diligent about taking care of what needs to be taken care of YOURSELF, you simply don't need to avail yourself of this "convenience" feature).
- If you DO want to take advantage of it, the basic approach is simple:

```
$ cat test_with.py
with open('best_goalies.txt', 'r') as f:
    goalie_names = f.read()
    print(goalie_names)
```

\$ python3 test\_with.py
Patrick Roy
Roberto Luongo
[etc]

5. <mark>I/O</mark>



https://upload.wikimedia.org/wikipedia/commons/3/33/Teletype-IMG\_7287.jpg

## Python3 Command Line I/O in general...

- Input may be:
  - Interactive (taken from the user's keyboard), like the following "guess the number" example on the next slide
  - "Piped in" from <stdin>
  - Read from a file, either hardcoded in the program or specified on the command line
- Output may be:
  - Sent to <stdout> (which may just be the console, or may be part of a pipeline), like our "guess the number" example
  - Written to a file, either hardcoded or specified on the command line
  - Output may also include error messages, which need to go to <stderr> (not <stdout>)
- Handling more sophisticated input and output properly requires taking the right approach

#### **Example of Simple Python3 Interactive Command Line Input/Output**

```
""" Guess A Number (Illustrating Basic Python3 I/O) """
from random import randint
print("Guess the number from 0 to 10")
target = randint(0, 10)
user guess = -1 # an impossible starting value
while user guess != target:
    try:
        user guess = int(input("What's your guess? "))
    except ValueError: # catch real numbers, alphanumerics
        print("Please enter a whole number from 0 to 10")
        continue # start a new interation of the loop
    if user guess < 0 or user guess > 10:
        print("Number's between 0 and 10. Try again.")
    elif user guess < target:</pre>
        print(user guess, " is too low. Try again.")
    elif user guess > target:
        print(user guess, " is too high. Try again.")
    else:
        print("You guessed it!")
```

#### Input: read from <stdin> <u>OR</u> from filenames supplied on the cmd line

The Un\*x model traditionally emphasized combining single purpose tools into command "pipelines." The "modern" temptation is to build standalone code that specializes in doing one thing from "soup to nuts" instead of staying true to the Un\*x "pipeline of tools" model. Using the fileinput library will let you read from <stdin> (or from filenames):

Beware: <stdin> can include awaiting keyboard-entered input unless you explicitly
 check to make sure stdin is NOT a TTY with
 import sys
 if len(sys.argv) > 1 or not sys.stdin.isatty():

**Reference:** https://docs.python.org/3/library/fileinput.html

## "Casual" Output with Print()

- The basic print command is pretty straightforward: print("something")
- If printing multiple variables (or literal strings), separate them with commas
- If you prefer to use plus signs to concatenate multiple variables (or literal strings), be sure all concatenated items are strings (or coerce them into becoming strings with str(x))
- If you want to print stuff without an automatic newline, use end=""

```
import time
print("Run started at:", time.asctime(time.gmtime()), end="")
time.sleep(10)
print(" ... Run ended at:", time.asctime(time.gmtime()))
```

## "What about Unicode?"

- It's tempting to assume that text will just be plain old ASCII text. That's not the case for Python3. Text in Python3 CAN and often WILL include Unicode characters.
- To FIND a Unicode character to add, see <a href="https://codepoints.net/planes">https://codepoints.net/planes</a>
- To display a Unicode character in Python, three "easy" ways to refer to them:

```
x1 = "\u270B \u2699"
x2 = "\N{RAISED HAND} \N{Gear}"
x3 = "\u2699"
x3 = "\u2699"
w"
print(x1) (or x2 or x3 - they'll all display the same)
print(f"{x3!a}") (Now Unicode characters will be shown escaped)
```

- Note that some Unicode characters may appear "small" in text output; there's no easy way to scale individual characters (it's still just text, right?). Fonts may also differ in the number of Unicode characters they know how to display. A userful page: https://www.alanwood.net/unicode/fonts\_macosx.html
- When opening a unicode text file, specify encoding="UTF-8" in the open statement

## Formatting Output with "f" strings

- If you need to produce a carefully formatted report in Python3, use "f" strings
- Assuming we've got a customer record as defined on slide 77, we could write:

```
import sys
if sys.version_info < (3,6):
    # pylint: disable-next=broad-exception-raised
    raise Exception("Must use Python 3.6 or later")
[...]
print(f"{x.name:30}")
print(f"{x.address:30}")
print(f"{x.city}, {x.state}, {x.zip}")
print(f"attn: {x.poc:30}")
```

Acme Widgets 123 Main St Springfield, Oregon, 97477 attn: John Smith

## "f" string formatting characters

- f"{variable\_name<modifiers>}"
- some modifiers:

: 8	a right justified field 8 chars wide (wider string? it will keep going and still be "fully shown")
:8.8	right justified field, truncated to 8 characters
:<8	a left justified field 8 chars wide
:.<20	add .'s to right pad a 20 char wide field
: ^8	a centered field 8 chars wide
:.2f	a float with two digits past the decimal point
:, :,.2f =	use commas as thousand seperators (can also use _ ) formatted with commas and two digits past the dec. show the variable name as part of the output

import datetime
print(f"{datetime.datetime.now()
:%Y-%m-%d %H:%M}")

## "Pretty Printing" Python3 Data Structures

• Use pprint instead of print:

from pprint import pprint

```
b_list = [{'name': 'Bob Smith',
    'email_address': 'bsmith@protonmail.com',
    'html_format_email': False},
    {'name': 'Jane Doe',
    'email_address': 'jane@apple.com',
    'html_format_email': True}]
```

```
for obs in b_list:
    pprint(obs, indent=4)
```

• See https://docs.python.org/3/library/pprint.html

## Using print for Output in Python3 Will Sometimes Trigger Warnings

• Many programs use print for output. Some linters may unexpectedly complain. For example, ruff (with some configs) will complain if it sees print in code:

```
$ ruff check helloworld.py
helloworld.py:2:1: T201 `print` found
Found 1 error.
```

• So WHAT'S going on there? See <a href="https://docs.astral.sh/ruff/rules/print/">https://docs.astral.sh/ruff/rules/print/</a> The assumptions seem to be: <a href="mailto:print">print</a> statements are often used for "hillbilly" debugging purposes, and thus <a href="mailto:print">print</a> statements may be "inappropriate" if seen in prod code. Thus <a href="mailto:print">print</a> statements should apparently either be replaced with logging routines, instead (or you could substitute <a href="mailto:sys.stdout.write">sys.stdout.write</a>) Sorry, no, not doing either of these things.

```
"""Everyone's favorite first program."""
import sys
```

sys.stdout.write("Hello, world!\n")

## Writing Logging Entries From Python3 On A Mac

\$ cat sample\_logging.py
""" Sample write to syslog """
import syslog

<mark>syslog.openlog</mark>("<mark>Python3</mark>") <mark>syslog.syslog</mark>(syslog.LOG\_ALERT, "<mark>The frozbaz flibberted</mark>")

```
$ python3 sample_logging.py
```

```
$ tail /var/log/system.log
[...]
Mar 23 17:45:10 JS Python3[32420]: The frozbaz flibberted
[...]
```

## What If I Want to Send ERROR Output To <stderr> (instead of a log)

Good tips can sometimes be found on StackOverflow and similar web programming forums. For example, https://stackoverflow.com/questions/5574702/how-do-i-print-tostderr-in-python has a nice solution for consistently writing error output to <stderr>

```
import os
import sys
from functools import partial
error = partial(print, file=sys.stderr)
```

```
myfilename='foobar-blah.txt'
```

```
# make sure the user-specified file exists
if not os.path.isfile(myfilename):
    error("\n***FATAL ERROR: " + myfilename + " doesn't exist")
    sys.exit(1)
```

## "Isn't assert 'good enough' for printing errors?"

• "Fans" of the assert statement might ask, "Why not just use assert?" import os

```
myfilename='foobar-blah.txt'
# make sure the user-specified file exists
assert os.path.isfile(myfilename), "myfilename doesn't exist"
```

- Answer:
  - assert is ONLY meant for debugging, and is NOT meant for production error handling
  - In fact, -O (the PYTHONOPTIMIZE flag) will often be set for production Python runs. If that's done, it will suppress assert checks altogether. Errors should be **explicitly** sent to stderr, perhaps using the sample code shown on the earlier slide instead.

#### 6. Simple Data Types: Integers, Floats, Strings, Booleans, etc.



Remember: Python3 strings can be of effectively unlimited length. Source: https://commons.wikimedia.org/wiki/File:TwineBallCawkerKs.jpg

```
f_{00} = -29005
print(type(foo))
<class 'int'>
bar = 384.21599
print(type(bar))
<class 'float'>
baz = "The quick brown fox"
print(type(baz))
<class 'str'>
within tolerance = True
print(type(within tolerance))
<class 'bool'>
buff = b'Some bytes blah'
                                            \leftarrow avoid using bytes whenever possible
print(type(buff))
<class 'bytes'>
str buff = str(buff, encoding='utf-8') \leftarrow convert bytes to str
print(type(str buff))
<class 'str'>
```

## Some Operations on Strings / String Methods



### Using Regular Expressions on Strings with the re Library

```
import re
some string = "The trees are greener in Oregon"
new string = re.sub("The trees are", "Everything is",
    some string)
print(new string)
Everything is greener in Oregon
another string = "MINNESOTA has many lakes"
found minnesota = re.search("Minnesota", another string,
    re.IGNORECASE)
if found minnesota:
    print("Minnesota found in test string")
Minnesota found in test string
```

• For more information, see <a href="https://docs.python.org/3/library/re.html">https://docs.python.org/3/howto/regex.html</a> ).

#### 7. LISTS: A Fundamental Python3 Data Structure

#### WHEN YOU START PROGRAMMING



**AFTER A WHILE** 

Source: https://www.pinterest.com/pin/290834088447075737/

## Python3 Lists

- Python3 offers basic simple ("atomic") data types such as integers, floats, and strings.
- Python3 ALSO offers several standard data structures, including lists, dictionaries, sets, tuples, and the ability to create user-defined custom data structures. Let's start with lists.

• Lists:

- Lists are **ordered** sets of comma-separate variables enclosed within **square bracket**s.
- List elements can be **any** data type (integers, strings, lists, etc.)
- Lists are changeable.
- Lists can have **duplicate entries**.
- Lists act as a substitute for an actual "array" data type.
- Many of Python3's most powerful (and most complicated/commonly misunderstood) commands operate on lists.

#### **Basics of Creating and Referring To Parts of Lists**

- a list = [0.8, 0.4, 0.2, 1.5, -3, 0.32, 0.01]
- print(a\_list[3]) ← 4th item in the list
  1.5
- print(a\_list[-2]) ← counting inward from the right hand side
  0.32
- print(a\_list[2:5]) ← starting and ending indicies (a "slice")
  [0.2, 1.5, -3]
- The above examples are numeric, but elements could have been strings, dictionaries, etc.

```
b_list = [{'name': 'Bob Smith',
    'email_address': 'bsmith@protonmail.com',
    'html_format_email': False},
    {'name': 'Jane Doe',
        'email_address': 'jane@apple.com',
        'html_format_email': True}]
for obs in b_list:
    print(obs)
```

## Some Approaches to Performing List "Surgery"



#### Two common errors when working with list functions

• Some fn's work on lists "in place" (top of the preceding slide); other fn's output a new list.

Functions may be offered in BOTH variants. Classic example -- sort() vs sorted(): sort() sorts an existing list in place. sorted() produces a new sorted list, leaving the original list intact "as-is."

See https://docs.python.org/3/howto/sorting.html

Your first thought <u>ANY</u> time you use <u>ANY</u> function to manipulate a list should be to confirm: "does the function I've selected work "in place" or does it produce a new list?"

Some functions add a new label pointing to a shared existing list (like a Un\*x file 'link').
 Other functions produce a new (and totally independent) copy of the original list.

This subtle difference is the different between a Python3 shallow copy and a deep copy.

See https://docs.python.org/3/library/copy.html

## zip: combine lists (if alpha/beta were sets, random pairs would form)

- Using zip to compose tuples from two lists: alpha = ["apples", "cherries", "peaches", "plums"] beta = ["green", "red", "pink", "purple"] fruit\_colors = zip(alpha, beta) for elements in fruit\_colors: print(elements) ('apples', 'green') ('cherries', 'red') ('peaches', 'pink') ('plums', 'purple')
- Prefer a dictionary? That's an option as output from zip, too:

```
fruit_colors_dictionary = dict(zip(alpha, beta))
print(fruit_colors_dictionary)
{'apples': 'green', 'cherries': 'red', 'peaches': 'pink',
'plums': 'purple'}
```

#### **Comprehensions: A Succinct Way of Operating on A List's Elements**

• List comprehension example:

```
newlist = []
oldlist = [0.8, 0.4, 0.2, 1.5, -3, 0.32, 0.01]
newlist = [x for x in oldlist if x < 0.5]
print(newlist)
[0.4, 0.2, -3, 0.32, 0.01]</pre>
```

• Yes, that's shorter, but is that less confusing/more easily understood than the following?

```
newlist = []
oldlist = [0.8, 0.4, 0.2, 1.5, -3, 0.32, 0.01]
for x in oldlist:
    if x < 0.5:
        newlist.append(x)
print(newlist)</pre>
```

Reference: https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

### 8. DICTIONARIES



https://commons.wikimedia.org/wiki/File:OED2\_volumes.jpg

## **Python3 Dictionaries**

• Dictionaries consist of key: value pairs, indexed by the key values. Dictionaries live within "curly braces:" { }

- Dictionary **keys** (left hand side values) MUST be unique. They can be any immutable type (quoted strings, integers, etc).
- Dictionary values (right hand side values) can also be strings, integers, etc. They do NOT need to be unique.
- New entries can be added to an existing dictionary, and existing dictionary entries can be updated. You'll likely find yourself using Python3 dictionaries a LOT when using Python3.
- JSON data will often be read into a Python3 dictionary with json.loads()
- A Python3 dictionary can be written out in JSON format with json.dumps()
   See https://docs.python.org/3/library/json.html

### Sample Code to Read and "Pretty Print" JSON Data

#### \$ cat m3aawg-extract.json1

```
{"count":364178,"time_first":"2013-08-20 01:41:16","time_last":
"2024-03-22 08:55:23","rrname":"m3aawg.org.","rrtype":"NS","bailiwick":
"m3aawg.org.","rdata":["udns1.ultradns.net.","udns2.ultradns.net."]}
```

{"count":214949,"time\_first":"2011-12-29 18:24:28","time\_last": "2024-03-22 09:52:38","rrname":"www.m3aawg.org.","rrtype":"CNAME", "bailiwick":"m3aawg.org.","rdata":["m3aawg.org."]}

#### \$ cat sample-json.py

"""demonstrate reading a JSON file and converting it to a dictionary""" import json from pprint import pprint

```
f = open("m3aawg-extract.jsonl", "r", encoding="utf-8")
results = f.readlines()
f.close()
```

```
for some_element in results:
    one_obs = json.loads(some_element)
    pprint(one_obs, indent=4)
```

### Sample Output

{

```
'bailiwick': 'm3aawg.org.',
'count': 364178,
'rdata': ['udns1.ultradns.net.', 'udns2.ultradns.net.'],
'rrname': 'm3aawg.org.',
'rrtype': 'NS',
'time first': '2013-08-20 01:41:16',
'time last': '2024-03-22 08:55:23'}
'bailiwick': 'm3aawg.org.',
'count': 214949,
'rdata': ['m3aawg.org.'],
'rrname': 'www.m3aawg.org.',
'rrtype': 'CNAME',
'time first': '2011-12-29 18:24:28',
'time last': '2024-03-22 09:52:38'}
```

#### A requested dictionary element MUST exist, or you'll get a KeyError

fruit\_prices = {"apples":1.52, "bannanas":0.52, "cherries":3.43, "grapes":1.84, "oranges":1.21, "peaches":1.72, "pears":1.59} print("cranberries", fruit\_prices["cranberries"])

**KeyError**: 'cranberries'

The prudent thing to do? Check to be sure the key you're interested in exists:

```
fruit_prices = {"apples":1.52, "bannanas":0.52, "cherries":3.43,
    "grapes":1.84, "oranges":1.21, "peaches":1.72, "pears":1.59}
if "pears" in fruit_prices:
    print ("pears", fruit_prices["pears"])
```

pears 1.59

## You can loop ("iterate") over a Python3 dictionary...

- Some entries in a Python3 dictionary may need to be interated over. You can do so over the keys, the values, or the key, value pairs. For the dictionary called dict --
- Iterating over keys:

```
for k in dict:
    [...]
```

• Iterating over values:

```
for v in dict.values():
    [...]
```

• Iterating over key, value pairs:

```
for k, v in dict.items():
    [...]
```
#### 9. Other Python3 Data Structures



https://commons.wikimedia.org/wiki/Category:Pantex\_Plant#/media/ File:Pantex\_Plant\_Magazine\_4-42.png

#### **Python3 Sets**

- Among other things, Python3 sets can be handy for accumulating unique values while iterating over a big bunch of values.
- Python3 sets are unordered.
- Sets (by definition) can't have duplicate elements.
- Like dictionaries, sets live within curly braces.

https://docs.python.org/3/tutorial/datastructures.html#sets

## **Working With Sets**

- Make an empty set with the set() function:
   my\_pets = set()
- Add elements to a set:

```
my_pets.add("Fido")
my_pets.update({"Fluffy", "Smokey", "Bootsy"})
print(my_pets)
{'Fido', 'Smokey', 'Fluffy', 'Bootsy'}
```

• Length of a set:

```
print(len(my_pets))
4
```

• Test to see if an entry is present:

```
if "Smokey" in my_pets:
    print("Smokey found in my_pets")
```

## **Python3 Tuples**

• Tuples are yet another built-in Python3 data structure. They're two or more values (integers, floats, strings, lists, ...) separated by commas (the parens are optional, but customary). The items in the tuple can be of different types.

$$x = (15, 8, 7, 12) y = ("ginger", "turmeric") z = ({5,8,12}, {10,8,7}, {-1, -3, -2})$$

• Tuples can be "unpacked:"

• individual elements and sub-elements can also be accessed from tuples:

```
print(z[1])
{10, 8, 7}
print(list(z[2])[1])
-1
```

#### dataclasses and a sample user-defined custom dataclass

```
""" demonstrate use of dataclass to create a sample class """
import sys
from dataclasses import dataclass
if sys.version info < (3,7):
    # pylint: disable-next=broad-exception-raised
    raise Exception ("Must use Python 3.7 or later")
@dataclass(kw only=True)
class CustomerRecord:
    """ customer record definition """
    name: str
    address: str
    city: str
    state: str
    zip: int
   phone: str
    poc: str
x = CustomerRecord(name="Acme Widgets", address="123 Main St",
    city="Springfield", state="Oregon", zip=97477, phone="555-343-1234",
    poc="John Smith")
print(x)
# example of accessing a single field from the object
print(x.phone)
```

#### **10.** Packages/Libraries

The role of 3<sup>rd</sup>-party packages/libraries

Installing 3<sup>rd</sup>-party packages

Virtual environments

What if there are multiple library options for a given task?

**Copyrights and Licensing** 

Security of 3<sup>rd</sup>-party packages/libraries



Different kind of library (but note book at start of right 2<sup>nd</sup> shelf) https://commons.wikimedia.org/wiki/File:SteacieLibrary.jpg

#### The Role of Packages/Libraries

- Much of the power of Python3 comes from a large portfolio of freely available packages/libraries
- These are normally distributed via PyPI (see https://pypi.org/). PyPi's HUGE:

522,350 projects 5,471,523 releases 10,531,117 files 793,833 users

• Python3 packages are simple to install, in most cases by just saying:

```
$ pip3 install <package_name>
```

• Before installing packages from PyPI, however, a few cautionary notes/recommendations

#### Use A Virtual Environment To Avoid Running Into "Package Conflicts"

- The process in detail? See <a href="https://docs.python.org/3/tutorial/venv.html">https://docs.python.org/3/tutorial/venv.html</a>
- In a nutshell, after you've used pip3 to install venv (only need to do that once):
  - \$ mkdir some\_project
    \$ cd some\_project
    \$ python3 -m venv .venv
    \$ source .venv/bin/activate
    [use or work on your project]
  - \$ deactivate
  - \$ **cd**
- Special considerations to keep in mind:
  - You'll need to install any packages you want to have in the virtual environment
  - Use pipx instead of pip when installing packages in a virtual environment (see https://pipx.pypa.io/stable/)
  - Upgrading packages in one virtual environment will NOT automatically upgrade the packages in another environment (of course, this is just as intended)
  - It can be potentially easy to lose track of the virtual environments you may have in use<sub>80</sub>

#### Choice of Packages: Pick <u>Really</u> Carefully!

- Python3 gives you many community—contributed options for doing common tasks (such as for accessing a RESTful API over the Internet, or for parsing JSON data)
- You may be tempted to just use the first package you find that seems to do what you need
- This can be a mistake. Packages will vary across many dimensions, including:
  - Ease-of-use (Is there documentation? can you find examples of using the code?)
  - Popularity (Good libraries tend to be popular and used by many users)
  - **Development status** (Was the library developed and released, but then abandoned? Are bugs still getting chased and squashed? Are there multiple active contributors?)
  - **Performance** (For example, there may be orders of magnitude difference in the performance of various JSON parsing libraries)
  - Security (If you're carefully crafting your own code so it's secure, don't undercut that security with a potentially woefully-insecure (or overtly hostile!) 3<sup>rd</sup> party library!)
- Check your favorite search engine for reviews, benchmarks and bakeoffs
- One objective starting clue: look at package dates and popularity.

#### libraries.io Stats For Two Packages Used for Making RESTful Queries

→ 🙆 🗘 🖞 https://libraries.io/pypi/pycurl	☆ 🤄 Search	$\checkmark$
pycurl	Stats	
Release 7.45.3	Dependencies	0
PycURL A Python Interface To The cURL library	Dependent packages	188
Homepage - PyPI	Dependent repositories	4.15K
	Total releases	56
Kayaworda	Latest release	25 days ago
curl libeurt urlib wast dewalead file transfer bttp www	First release	Apr 6, 2005
License	SourceRank	15

	🗉 🔂 🧐 Search 🖳	A https://libraries.io/pypi/requests
	Stats	quests
0	Dependencies	lease 0.0.1
60.2K	Dependent packages	on HTTP for Humans.
152K	Dependent repositories	iepage - Repository - PyPI - Python
151	Total releases	
Jul 20, 2023	Latest release	words
Feb 14, 2011	First release	t cookies forhumans http://umans.ovthon.ovthon-requests.requests
51K	Stars	
9.33K	Forks	verse
1,337	Watchers	
615	Contributors	Stall
12.3 MB	Repository size	pip install requests <del>==0.0.1</del>
31	SourceRank	
	SourceRank	

Either of these would be a worthy choice...

But avoid "similarly-named" (but "ancient," "brand new," or "little used") packages...

#### The "most relevant" package may not be the one you actually want

requests	Q	Help Sponsors Log in Register
Filter by <u>classifier</u>	10,000+ projects for "requests"	haha Order by Relevance ~
<ul> <li>Framework</li> <li>Topic</li> <li>Development Status</li> </ul>	requests2 2.16.0 Python HTTP for Humans.	May 26, 2017
<ul> <li>License this&gt;</li> <li>Programming Language</li> </ul>	requests 2.31.0 Python HTTP for Humans.	May 22, 2023
<ul><li>Operating System</li><li>Environment</li></ul>	requests5 1.0.0 无与伦比的简单且强大的requests	Apr 20, 2020
<ul> <li>Intended Audience</li> <li>Natural Language</li> </ul>	requests3 0.0.0 Name Squatting.	Mar 16, 2018
Typing	batch-requests 0.1.2 make batch http requests	Jun 20, 2022
	requests_spider 0.0.8         Web crawling framework like flask.	Sep 11, 2018

#### "So I Think I'd Like To Upgrade ALL My Python3 Packages..."

• People (understandably) may want to keep their Python3 packages up to date. This can sometimes be "tricky." A common approach to try doing that looks like:

```
[check there isn't already a requirements.txt file in this dir then...]
$ pip3 freeze > requirements.txt
[review the list in requirements.txt before continuing]
$ pip3 install --upgrade -r requirements.txt
```

#### • Didn't use virtual environments? You MAY encounter tricky-to-resolve conflicts. Example:

INFO: pip is looking at multiple versions of argcmdr to determine which version is compatible with other requirements. This could take a while.

```
ERROR: Cannot install -r requirements.txt (line 14), argcmdr==1.0.1 and argcomplete==3.1.6 because these package versions have conflicting dependencies.
```

```
The conflict is caused by:

The user requested argcomplete==3.1.6

argcmdr 1.0.1 depends on argcomplete<3 and >=1.9.4

[etc]
```

#### "So What Do I Do If I'm Trapped In Some Sort of Dependency Hell?"

- Reconsider: do you **REALLY** need to upgrade all your packages? Are you **SURE?**
- If so, the first step is to learn more about the packages that rely on the conflicting modules. A great tool for that: https://github.com/tox-dev/pipdeptree Once you've installed pipdeptree, try:

\$ pipdeptree -r

• Now that you can see what conflicts exist, internalize the classic Stones song "You Can't Always Get What You Want," see https://www.youtube.com/watch?v=ZUqSNbJuGOw

After you've finished listening to that immortal song, begin by deciding what conflicting packages you're going to remove. Remove the first of those with pip3 uninstall

Now rerun pipdeptree -r and resolve the next conflict that still exists. Iterate until done. Attempt to update again (if you still want to try to do this).

#### **Python Package Copyright and Licensing**

• Each package on <a href="https://pypi.org/">https://pypi.org/</a> has a license summary. Some examples include:

Meta	Meta	Meta	Meta
License: BSD	License: MIT License	License License: GPLv2	License: Apache Software License
			(Apache Software License 2.0)

 Be SURE you know which license(s) apply to the libraries/packages you're using! If you installed the handy pipdeptree package mentioned a slide or two back, try:

- \$ pipdeptree --license
- You should ALSO be sure you know your rights and obligations under those licenses! A good starting point: https://opensource.org/licenses
- Trying to pick a license for YOUR OWN code? Talk to your management and company legal counsel. For a private project, maybe see https://choosealicense.com/

"Can I TRUST the packages/libraries I download from PyPI?"

One analysis from circa 2021...

🗐 110% 😭 🧐 Search

 $\mathbf{F}$ 

#### The **A** Register<sup>®</sup>

# About half of Python libraries in PyPI may have security issues, boffins say

Coding lingo's community says it has a plan to mitigate supply chain vulnerabilities, though

A Thomas Claburn

Wed 28 Jul 2021 // 22:14 UTC

Boffins in Finland have scanned the open-source software libraries in the Python Package Index, better known as PyPI, for security issues and said they found that nearly half contain problematic or potentially exploitable code.

In <u>a research paper</u> distributed via ArXiv, Jukka Ruohonen, Kalle Hjerppe, and Kalle Rindell from the University of Turku describe how they subjected some 197,000 Python packages available through PyPI to a static analysis tool called Bandit and found more than 749,000 instances of at best poor, or at worst insecure, programming.

"Even under the constraints imposed by static analysis, the results indicate [the] prevalence of security issues; at least one issue is present for about 46 per cent of the Python packages," the researchers claimed.

Of the issues identified, most (442,373) are low severity. About 227,426 are moderate severity, present in about 25 per cent of PyPI packages. And about 80,065 are high severity, present in about 11 per cent of PyPI packages.

#### Running The Same Static Code Checker Those Researchers Used...

Sample Bandit run (https://github.com/PyCQA/bandit) on the Python requests library:

```
$ bandit -r requests-2.31.0 -n 3 -111
[...]
Test results:
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider
usedforsecurity=False
   Severity: High
                 Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
  More Info: https://bandit.readthedocs.io/en/1.7.9.dev1/plugins/b324 hashlib.html
  Location: requests-2.31.0/requests/auth.py:148:23
                          x = x.encode("utf-8")
147
                      return hashlib.md5(x).hexdigest()
148
[...]
                                                        Are those flaws a "big
Run metrics:
                                                        deal?" YOU'LL have to
      Total issues (by severity):
             Undefined: 0
             Low: 532
                                                        review them & decide
             Medium: 116
             High: 3
                                                        for yourself...
```

## And A Recent Example of a Supply Chain Attack...

# Top Python Developers Hacked in Sophisticated Supply Chain Attack

Multiple Python developers get infected after downloading malware-packed clone of the popular tool Colorama.



Multiple Python developers, including a maintainer of Top.gg, were infected with information-stealing malware after downloading a malicious clone of a highly

#### popular tool, Checkmarx reports.

Called Colorama, the utility makes ANSI escape character sequences work on Windows and has more than 150 million monthly downloads.

To mount their supply chain attack, the hackers cloned the tool, inserted malicious code into it, and placed the

Ref: https://www.securityweek.com/top-python-developers-hacked-in-sophisticated-supply-chain-attack/ and https://arstechnica.com/gadgets/2021/03/more-top-tier-companies-targeted-by-new-type-of-potentially-serious-attack/

# 11: argparse: a built-in library that simplifies documenting your command-line interface



https://xkcd.com/1168/

#### Producing An In-Program Help Synopsis with argparse

 In addition to other program documentation, it can be helpful to be able to get a brief command summary in your program. Use argparse to do that automatically:

```
import argparse
parser = argparse.ArgumentParser(description="Plot lat, long, count points.",
                                 allow abbrev=False)
parser.add argument ('-v', '--version', action='store true', dest='version')
parser.add argument("--myfilename", type=str, action="store")
# going to let the user pick one (and only one) geographical extent
group = parser.add mutually exclusive group()
group.add argument ('--world', '-w', action="store true")
group.add argument ('--US', '--united states', '--us',
                   action="store true")
group.add argument ('--ca', '--cal', '--california', action="store true")
group.add argument ('--fl', '--fla', '--florida', action="store true")
# "or" would run into Python reserved words
group.add argument ('--ore', '--oregon', action="store true")
group.add argument('--tx', '--tex', '--texas', action="store true")
args = parser.parse args()
```

• You then need to grab the bits you want from those parsed arguments...

#### **Accessing Arguments from Argparse**

```
if args.version:
    print("plot_lat_long Version 1.0")
    sys.exit(0)
```

```
myfilename = args.myfilename
```

```
if args.world:
    [...]
elif args.US:
    [...]
elif args.ca:
    [...]
elif args.fl:
    [...]
elif args.ore:
    [...]
elif args.tx:
    [...]
```

#### **Sample Automatically Generated Help Summary**

Plot lat, long, count points.

options:

```
-h, --help show this help message and exit
-v, --version
--myfilename MYFILENAME
--world, -w
--US, --united_states, --usa, --us
--ca, --cal, --california
--fl, --fla, --florida
--ore, --oregon
--tx, --tex, --texas
```

#### 12. Sanitizing/Validating/Encoding Inputs/Fuzzing

HI, THIS IS WELL, WE'VE LOST THIS OH, DEAR - DID HE DID YOU REALLY YOUR SON'S SCHOOL. BREAK SOMETHING? YEAR'S STUDENT RECORDS. NAME YOUR SON WE'RE HAVING SOME Robert'); DROP I HOPE YOU'RE HAPPY. IN A WAY-COMPUTER TROUBLE. TABLE Students; -- ? AND I HOPE - OH, YES. LITTLE YOU'VE LEARNED BOBBY TABLES, TO SANITIZE YOUR WE CALL HIM. DATABASE INPUTS.

https://xkcd.com/327/

#### **Defensive Coding: Sanitizing/Validating/Encoding Input**

- If you are developing an application for general use, assume that users WILL accidentally (or intentionally) throw unexpected/malicious/random input at it. As some people put it, "expect cats to dance on the user's keyboard while the program is running."
- It is UP TO YOU to ensure that you don't inadvertently allow unexpected input to impact your system, your code, or the data your program interacts with. There are different approaches to handling this issue, with potentially different levels of protection:
  - Static typing (perhaps using my[py])
  - Full input validation (perhaps using pydantic, building on static typing)
  - Escaping all HTML-format input
  - Using specially tailored validators (for example, for email addresses)
- Think you've got it all handled? Try using a **fuzzer** to test it (the bad guys certainly will!)

#### Static type declarations for Python: my[py]

• https://mypy.readthedocs.io/en/stable/getting\_started.html illustrates the key concept:

```
# The "name: str" annotation says that the "name" argument should be a string
# The "-> str" annotation says that "greeting" will return a string
def greeting(name: str) -> str:
    return 'Hello ' + name
```

This function is now *statically typed*: mypy will use the provided type hints to detect incorrect use of the greeting function and incorrect use of variables within the greeting function. For example:

```
def greeting(name: str) -> str:
    return 'Hello ' + name

greeting(3)  # Argument 1 to "greeting" has incompatible type "int"; expected "str"
greeting(b'Alice') # Argument 1 to "greeting" has incompatible type "bytes"; expected "str"
greeting("World!") # No error

def bad_greeting(name: str) -> str:
    return 'Hello ' * name # Unsupported operand types for * ("str" and "str")
```

#### "So How About Just Defensively Escaping Any HTML Input?"

• If HTML input must be allowed, one (not recommended) approach is to escape the input:

```
import html
input = '<LI><B><A HREF="https://www.stsauver.com/joe/">Joe''s Site</A></B>'
sanitized = html.escape(input)
print(sanitized)
```

<LI&gt;&lt;B&gt;&lt;A HREF=&quot;https://www.stsauver.com/joe/&quot;&gt;Joes Site</A&gt;&lt;/B&gt;

• BUT note that https://cheatsheetseries.owasp.org/cheatsheets/ SQL\_Injection\_Prevention\_Cheat\_Sheet.html explicitly says:

#### "Defense Option 4: STRONGLY DISCOURAGED: Escaping All User-Supplied Input

In this approach, the developer will escape all user input before putting it in a query. It is very database specific in its implementation. This methodology is frail compared to other defenses and we CANNOT guarantee that this option will prevent all SQL injections in all situations."

#### Email address validation: https://pypi.org/project/email-validator/

The M3AAWG audience may be particularly interested in validating email addresses. This is a nice example where a special validating library may be a convenient option:

```
from email_validator import validate_email, EmailNotValidError
test_addresses = ['root@co.uk', 'jane@@blah']
for addr in test_addresses:
    try:
        emailinfo = validate_email(addr, check_deliverability=True)
        norm_email = emailinfo.normalized
        print("superficially valid: ", norm_email)
    except EmailNotValidError as e:
        print(str(e))
```

The domain name co.uk does not accept email. The email address is not valid. It must have exactly one @-sign.

#### Confident You're Set? Fuzz It! (Test Your Code With Random Input)

• See https://www.fuzzingbook.org/

This is a freely available and wonderful online book that has been written by Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. They say:

"You can use this book in four ways: [...] You can read chapters in your browser. Check out the list of chapters in the menu above, or start right away with the introduction to testing or the introduction to fuzzing. All code is available for download. [continues]"

"This work is designed as a *textbook* for a course in software testing or security testing; as *supplementary material* in a software testing, security testing, or software engineering course; and as a *resource for software developers*. We cover random fuzzing, mutation-based fuzzing, grammar-based test generation, symbolic testing, and much more, illustrating all techniques with code examples that you can try out yourself."

"When formatted as a PDF, @TheFuzzingBook now has 1,477 pages"

#### **13. Other Python3 Testing**



https://pixabay.com/photos/microscope-slide-research-close-up-275984/

#### Output Testing: Let's Look At A Python Uniform Random Number Generator – Does It Seem to Produce Uniformly Distributed Values?

```
$ cat ran0.py
```

```
#!/usr/local/bin/python3
""" from Numerical Recipes Ch7 """
def ran0(idum):
    IA = int(16807)
    IM = int(2147483647)
    IQ = int(127773)
    IR = int(2836)
   MASK = int(123459876)
    AM = float(1./IM)
    idum=idum^MASK
    k = int(idum/IQ)
    idum=int(IA*(idum-k*IQ)-IR*k)
    if idum < 0:
        idum=int(idum+IM)
    rand0=float(AM*idum)
    idum=int(idum^MASK)
```

```
return (rand0, idum)
(rand0, newseed)=ran0(int(77717))
for x in range(1,10000000):
   (rand0, newseed)=ran0(newseed)
   print(rand0)
```

```
$ python3 ran0.py > unif-dev.txt
```

```
$ cat test_unif-dev.py
#!/usr/local/bin/python3
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("unif-dev.txt")
ax = sns.displot(df)
plt.savefig("distro.pdf",dpi=300)
```

#### Is the output approximately uniformly distributed?



← uniformly distributed (yes, with a little "random ripple")

That's just "scratching the surface," however, see also https://en.wikipedia.org/wiki/Diehard\_tests and https://en.wikipedia.org/wiki/TestU01

#### Unit Testing with pytest ( https://docs.pytest.org/en/8.0.x/ )

• Simple rule for integrating testing into your code: anytime you write a new Python function, write one or more unit tests of that function at the same time. Trivial example:

```
$ cat my fruit pricing.py
def fruit pricing(fruit):
    fruit prices = {"apples":1.52, "bannanas":0.52, "cherries":3.43,
        "grapes":1.84, "oranges":1.21, "peaches":1.72, "pears":1.59}
    if fruit in fruit prices:
        return fruit prices[fruit]
    else:
        return None
def test pricing():
    assert fruit pricing("apples") == 1.52
    assert fruit pricing("bannanas") == 0.52
    assert fruit pricing("cherries") == 3.43
    assert fruit pricing("mangoes") == None
```

```
$ pytest my_fruit_pricing.py
```

#### Black Box Testing with https://pypi.org/project/pbbt/

- Black box testing assumes no knowledge of how a program works, it's just treated as, well, a "black box": inputs go into the black box, stuff happens, and outputs come out. It's often used to ensure that:
  - After new code is added, the program still runs as expected
  - Output remains consistent: for a given set of inputs, the same results get returned
  - All platforms perform the same: for instance, running the program on either Linux or BSD operating systems still returns the same results.
- In a nutshell, all those tests compare how things worked "before" with how things work "now," looking for changes. The assumption is that variability is generally unwanted (or at least something which should be carefully understood). This is admittedly a simplistic testing approach, but a good foundation on which more sophisticated tests can be added.
- You'll sometimes hear this referred to as "regression testing."
- See https://pypi.org/project/pbbt/ or l've got a full article about this at
   https://www.domaintools.com/resources/blog/black-box-testing-with the-python-black-box-tool-pbbt/

### **Profiling** Python3 Code

• If you're trying to speed up "long-running" (slow) code, the first question is "where does the program spend most of its time?" The easiest way to answer that question is with pyinstrument (see https://pyinstrument.readthedoc3s.io/)

\$ pyinstrument --from-path ./my\_python\_script.py

```
[any regular program output]
```

```
.____/___/ Recorded: 16:09:30 Samples: 19962
/_//_///_///_///_///_//___//_
Duration: 51.574 CPU time: 51.135
v4.6.1
```

Program: [program location here]

```
51.558 <module> <my_python_script>.py:1

    51.172 open pikepdf/_methods.py:342

    [2 frames hidden] pikepdf, <built-in>

    51.166 PyCapsule._open <built-in>

[etc]
```

#### **14. Exercises**



"U.S. Navy sailors lift a large log dubbed **Old Misery** during log physical training at the Naval Special Warfare Center at Naval Amphibious Base Coronado, Calif., on Feb. 3, 2009. The sailors are undergoing basic underwater demolition/SEAL training." [emphasis added] https://commons.wikimedia.org/wiki/File:Defense.gov\_News\_Photo\_090203-N-7303M-011.jpg

#### Why Exercises?

- You can read about/hear someone talk about a programming language, but you really won't understand that language until you actually struggle with it and try to use it to do something.
- Working on code will help improve your skills and make you a stronger coder.
- Some people find it hard to come up with sample programming projects. If you CAN come up with your own projects, that's great, work on those. If not, you can at least try some of these.
- Looking for sample solutions for these? See https://www.stsauver.com/joe/solutions/

#### **#1: Tic Tac Toe**

- Human is X, computer is O.
- Show a basic tic tac toe display and request a move from the human player.
- Update the "display." Now have the computer pick a move.
- Loop until three the same in a column (or row, or diagonal), or no one can win (it's tie).

#### Notes:

- Use a Python3 data structure (set, list, dictionary, tuple) as part of your code. For example, perhaps you want to have a set listing the available squares, a set listing the squares X has selected, and a set listing the squares O has selected. Or maybe you'd like to think about the board as a dictionary, and assign " " (free), "X", or "O" to each dictionary element?
- The core of this exercise is the computer's "move selector." Simplest approach is to pick a random unoccupied square, but you can do better than that, right? For example, if there's a winning square, always pick that. If there's NOT a winning square, but the human has two in a row, the computer should try to keep the human from winning (if it can).
- Remember, a sample solution is available at https://www.stsauver.com/joe/solutions/
## **#2: Recover Access to a PDF With A Forgotten Password**

- You created and saved an important PDF file. You decided it should have a password.
- Unfortunately, you forgot to make a note of the precise password you used. While you don't remember the exact password, you DO remember that it used a short potentially mixed-case alphanumeric password, at most six characters long.
- Write Python3 code to try opening the document with all possible short passwords.

## • A Few Hints:

- Use a Python3 library that's able to open a password protected PDF, and which will accept a programmatically-supplied password.
- ONLY try this on one of your OWN PDFs, stored LOCALLY on the system you're using. Do NOT attempt to find the password for a remote document that someone else owns. Need a passworded PDF? Download https://www.stsauver.com/joe/managers-amendment/

## **#3: Computational Linguistics: Let's Analyze Some Executive Orders!**

- The American Presidency Project maintains an archive of nearly a quarter million Presidential speeches and other Presidential remarks at https://www.presidency.ucsb.edu/documents
  Pick a category of documents (such as Executive Orders) and save at least half a dozen samples for each of at least half a dozen leaders (so 36 or more total files).
- Are those speechs written for a 12th grade audience? A college-level audience? Find a Python3 library that can help you assess this. Report the Flesch-Kincaid Grade Level plus at least one other reading measure of your choice for the half dozen leaders. For example:

george_bush	flesch=18.3	text_standard=17.0	doc_count=7
clinton	flesch=18.4	text_standard=17.0	doc_count=21
trump	flesch=21.0	text_standard=18.6	doc_count=21
obama	flesch=21.3	text_standard=28.0	doc_count=21
george_w_bush	flesch=24.1	<pre>text_standard=15.0</pre>	doc_count=22
biden	flesch=24.3	text_standard=17.0	doc_count=12

## #4: Make a Basic GUI Calculator

So far we've only talked about "command line" Python3. You may be wondering if Python3 can be used to make an actual graphic application – it can!

We haven't talked AT ALL about a graphic user interface for Python3, but they DO exist. For example, check out https://docs.python.org/3/library/tkinter.html

If you're so inclined, try creating a basic graphical user interface (GUI) point-and-click calculator.

If that's too daunting, become familiar with Tkinter by extending an already functioning sample calculator – see https://www.stsauver.com/joe/solutions/exercise-four/

Maybe create and add "memory" keys M+, M-, M Recall, M Exchange (swap current value and the memory value), M Clear, in a new vertical column to the left of the current keys?

