# CS50P W4 - Libraries

## 

## Random

The **random** module allows us to generate pseudo-random numbers for various distributions. https://docs.python.org/3/library/random.html

1. **choice()**

**import** random

coin = random.choice([“heads”, “tails”])

print(coin)

\*import random will import the whole module with all of its built-in functions. The choice() function found in the random module takes a sequence (a list here) and returns a random element.

**from** random **import** choice

coin = choice([“heads”, “tails”])

print(coin)

\*from random import choice will only import the choice() function. We do not have to specify random.choice().

1. **randint()**

**import** random

number = random.randint(1, 10)

print(number)

\*The randint() function found in the random module takes two integers as arguments and returns a random integer between those two integers.

1. **shuffle()**

**import** random

cards = [“jack”, “queen”, “king”]

random.shuffle(cards)

**for** card **in** cards:

print(cards)

\*The shuffle() function found in the random module will shuffle a list into a random order. It does not return anything; it directly shuffles the elements of a list.

1. **choices()**

**import** random

cards = [“jack”, “queen”, “king”]

print(random.choice**s**(cards, k=2))

\*The choices() function (plural) takes two arguments. The first is the population sequence cards and the second is k=2 specifies the number of items to return. This code will return a list of two randomly chosen elements.

1. Assigning **weights[ ]** to the elements

**import** random

cards = [“jack”, “queen”, “king”]

print(random.choice**s**(cards, weights[75, 25, 5], k=2))

\*The weights[ ] argument allows us to specify a sequence of weights that will be respected when drawing the cards. The numbers represent the percentage of time that each card will be drawn. The sequence must follow the same indexing as the population sequence and sum to 100.

\*In the code above, 75% of the time we will get a “”jack”, 25% of the time a “queen”, and 5% of the time a “”king”.

1. Debugging with **seed()**

**import** random

cards = [“jack”, “queen”, “king”]

random.seed(**0**)

print(random.choice**s**(cards, k=2))

\*The random.seed(0) function initializes the random number generator with the specific value 0. Seeding the random number generator ensures that we get the same sequence of random choices every time we run the program.

\*This is useful for debugging , testing or a situation where we need reproducible results.

1. Sampling without replacement **sample()**

**import** random

cards = [“jack”, “queen”, “king”]

print(random.sample(cards, k=2))

\*The sample() function takes the same two arguments but returns a list of **unique** elements chosen from the population sequence. Cards will not repeat in the result.

## 

## Statistics

The **statistics** module can be used to import functions to calculate means, medians, modes or other aspects of a data set. https://docs.python.org/3/library/statistics.html

**import** statistics

print(statistics.mean([100,90]))

\*The mean() function in the statistics module takes a list of values and returns the average of those two values.

## 

## Command-line arguments

The **sys** module allows us to take arguments at the **command line**. <https://docs.python.org/3/library/sys.html>.

1. **sys.argv[ ]**

**import** sys

print("hello, my name is", sys.argv[1])

\*The argv[ ] function found in the sys module allows us to learn about what the user typed in at the command-line.

\*argv[0] is the name of the file, argv[1] is the first argument, argv[2] the second, etc.

1. **try and except**

**import** sys

**try**:

print("hello, my name is", sys.argv[1])

**except** IndexError:

print(“Too few arguments”)

\*If the user does not enter the required arguments, we might get an IndexError (list index out of range). We can use a try and except block to anticipate such error.

**except** IndexError as e:

print(“Too few arguments”)

\*Using **as e** in a try and except block will bind the exception that was raised to the variable **e**, which allows access and manipulate the exception object within the except block.

1. **Conditionals** + **len()**

**import** sys

**if** len(sys.argv) **<** **2**:

print(“Too few arguments”)

**elif** len(sys.argv) **> 2**:

print(“Too many arguments”)

**else**:

print("hello, my name is", sys.argv[1])

\*We can use conditionals to add more guidance for the user to enter the right format.

1. **sys.exit()**

**import** sys

**if** len(sys.argv) **<** **2**:

sys.exit(“Too few arguments”)

**elif** len(sys.argv) **> 2**:

sys.exit(“Too many arguments”)

print("hello, my name is", sys.argv[1])

\*It is always recommended to separate the error handling from the rest of the code. The exit() function in the sys module allows us to exit the program if an error was introduced by the user.

1. Print all name tags with **no upper argv limit** using a **for loop**

**import** sys

**if** len(sys.argv) **<** **2**:

sys.exit(“Too few arguments”)

**for** arg **in** sys.argv:

print("hello, my name is", arg)

\*This version of the code will allow us to print all of the command-line arguments that the user enters. The only issue here will be that the program will also print argv[**0**] which stores the program name (name.py).

1. **slice()**

**import** sys

**if** len(sys.argv) **<** **2**:

sys.exit(“Too few arguments”)

**for** arg **in** sys.argv[**1:**]:

print("hello, my name is", arg)

\***slice()** will allow us to take a list and tell the compiler where we want it to consider the start of the list and the end of the list. sys.argv[**1:**].

\*Rather than starting at argv[**0**] (name.py), the compiler will start at the next element argv[**1**].

\*sys.argv[**1:-1**] will start at argv[**1**] and end at argv[ **-** **1**] excluding the last command-line argument.

## 

## Packages

Packages are **third party libraries** implemented as a **folder** that add functionality. The Python package manager **pip** allows you to install packages quickly onto the system.

1. `**pip install cowsay**`

**import** cowsay

**import** sys

**if** len(sys.argv) == **2**:

cowsay.cow(“hello, “ + sys.argv[**1**])

\*First we have to install the package with “pip install <package>” then import it like we would any library.

## 

## APIs

**API**s “**Application program interfaces**” allow you to connect to the code of others hosted on a server.

**requests** is a package that allows your program to make Web requests as a web browser would.

1. `**pip install requests**`

<https://itunes.apple.com/search?entity=song&limit=1&term=weezer>

\*Apple iTunes has its own API that we can access. This link is created specifically for a specific query reading Apple’s API documentation.

\*Notice that this query is looking for a song, with a limit of 1 result, that relates to the term called weezer.

\*Following the link will download a text file in JSON “JavaScript Object Notation” format, a text-based format used to exchange text-based data between applications.

{

"resultCount":1,

"results": [

{"wrapperType":"track", "kind":"song", "artistId":115234, "collectionId":1440878798, "trackId":1440879325, "artistName":"Weezer", "collectionName":"Weezer", "trackName":"Buddy Holly", "collectionCensoredName":"Weezer", "trackCensoredName":"Buddy Holly", "artistViewUrl":"https://music.apple.com/us/artist/weezer/115234?uo=4", "collectionViewUrl":"https://music.apple.com/us/album/buddy-holly/1440878798?i=1440879325&uo=4", "trackViewUrl":"https://music.apple.com/us/album/buddy-holly/1440878798?i=1440879325&uo=4",

"previewUrl":"https://audio-ssl.itunes.apple.com/itunes-assets/AudioPreview211/v4/b1/35/53/b13553c8-22f3-3e62-47cc-beaf65440f0e/mzaf\_9734530910938773283.plus.aac.p.m4a", "artworkUrl30":"https://is1-ssl.mzstatic.com/image/thumb/Music221/v4/d0/16/da/d016da24-577e-b584-3a5a-116efb5ca362/16UMGIM52971.rgb.jpg/30x30bb.jpg", "artworkUrl60":"https://is1-ssl.mzstatic.com/image/thumb/Music221/v4/d0/16/da/d016da24-577e-b584-3a5a-116efb5ca362/16UMGIM52971.rgb.jpg/60x60bb.jpg", "artworkUrl100":"https://is1-ssl.mzstatic.com/image/thumb/Music221/v4/d0/16/da/d016da24-577e-b584-3a5a-116efb5ca362/16UMGIM52971.rgb.jpg/100x100bb.jpg", "collectionPrice":10.99, "trackPrice":1.29, "releaseDate":"1994-02-28T12:00:00Z", "collectionExplicitness":"notExplicit", "trackExplicitness":"notExplicit", "discCount":1, "discNumber":1, "trackCount":10, "trackNumber":4, "trackTimeMillis":159587, "country":"USA", "currency":"USD", "primaryGenreName":"Pop", "isStreamable":true}]

}

**import** requests

**import** sys

**if** len(sys.argv) != **2**:

sys.exit()

response **=** requests.get("https://itunes.apple.com/search?entity=song&limit=1&term=" + sys.argv[**1**])

print(response.json())

\*requests.get() will use the API link, sys.argv[1] will allow the user to specify the artist at the command-line and return a JSON file that will be stored in the response variable.

\*print.json() will print the JSON text file standardized as a Python dictionary.

1. **json** library

**import** json

**import** requests

**import** sys

**if** len(sys.argv) != **2**:

sys.exit()

response **=** requests.get("https://itunes.apple.com/search?entity=song&limit=1&term=" + sys.argv[**1**])

print(json.dumps(response.json(), indent=2))

\*json.dumps() will take the response,json() and convert it to a more readable format. dumps() can take the additional parameter indent=2 to indent two spaces for additional readability.

1. Printing only the **trackName**

**import** json

**import** requests

**import** sys

**if** len(sys.argv) != **2**:

sys.exit()

response **=** requests.get("https://itunes.apple.com/search?entity=song&limit=50&term=" + sys.argv[**1**])

o **=** response.json()

**for** result **in** o[“results”]:

print(result[“trackName”])

\*In this version of the code we increased the song limit to 50 elements.

\*We store the result of response.json() object in a variable called **o**. Then we iterate using a for loop through the results in o[“results”] and print each result[“trackName”]

## Making Our Own Libraries

In a scenario where we may want to **re-use** some of our code repeatedly or to **share** with others, we can create our own **package**.

1. **sayings.py**

**def** main():

hello("world")

goodbye("world")

**def** hello(name):

print(**f**"hello, **{**name**}**")

**def** goodbye(name):

print(**f**"goodbye, **{**name**}**")

main()

1. **say.py**

**import** sys

**from** sayings **import** hello

**if** len(sys.argv) == **2**:

hello(sys.argv[**1**])

\*In the say.py file, even if we are only importing the hello() function (from sayings import hello) from our sayings.py file, the program will read the sayings.py file from top to bottom. Since we are calling main() at the bottom of saying.py, the program will not only import the hello() function but also **run** the main() function.

## \_\_name\_\_

\_ \_name\_ \_ is a special variable automatically set by Python. When we run a file from the command line \_\_name\_\_ is set to “\_\_main\_\_”. This allows us to **only** call **main()** if we run the file **directly**, rather than when it is imported as a module.

1. **sayings.py**

**def** main():

hello("world")

goodbye("world")

**def** hello(name):

print(**f**"hello, **{**name**}**")

**def** goodbye(name):

print(**f**"goodbye, **{**name**}**")

**if** \_\_name\_\_ **==** “\_\_main\_\_”:

main()

1. **say.py**

**import** sys

**from** sayings **import** hello

**if** len(sys.argv) == **2**:

hello(sys.argv[**1**])

\*By wrapping the call to main() in the conditional if \_\_name\_\_ == “\_\_main\_\_”:, we ensure that running say.py will ignore the call to main() in sayings.py, because \_\_name\_\_ in sayings.py will not be “\_\_main\_\_” when it is imported.

## PEP 8

A **style** [**guide**](https://peps.python.org/pep-0008/) that tries to standardize how Python code is written. Indentation - Tabs and spaces - Maximum line length - Blank lines - Imports. There are several programs that statically analyze the code and identify inconsistencies or mistakes.

**pylint**  - `**pip install pylint**` **pylint** <filename.py>

**pycodestyle** - `**pip install pycodestyle**` **pycodestyle** <filename.py>

**black** - `**pip install black**` : **black** <filename.py>