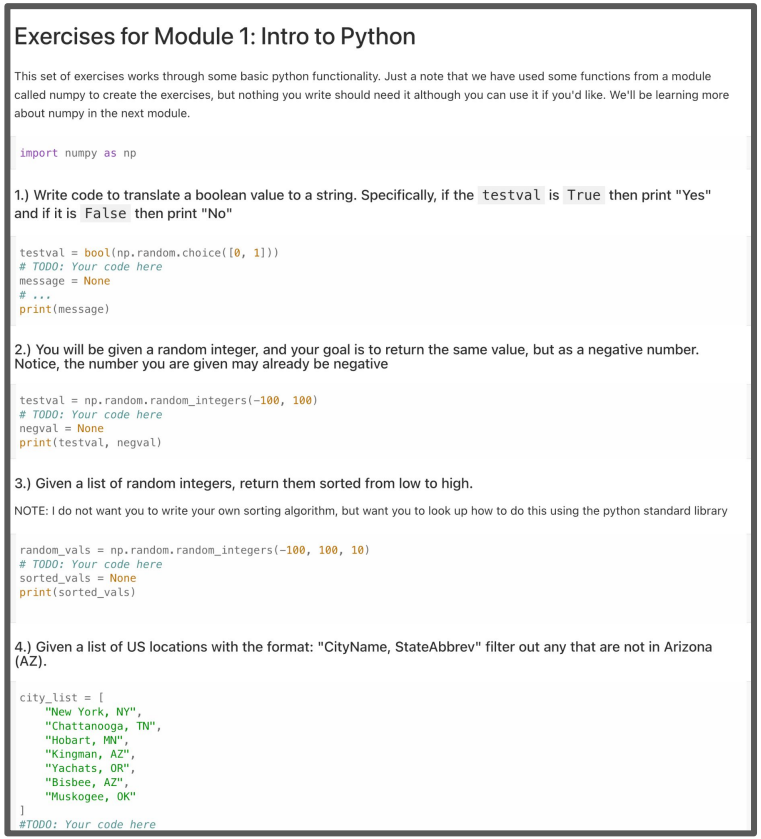
HAS Tools:

Functions & module imports

September 9, 2024

Your second assignment:

- The there is an assignment posted on D2L, but as before, all your work will be done on GitHub
- Homework notebook that you will modify is in `homework_submissions/hw2_python_exercises.ipynb`
- 10 points overall - 1 point for correctness of each answer and 3 points for general completion.
- No action/submission needed on D2L - I can see when you made commits/pushes on GitHub directly.
- We will have in class time towards the end of the day for getting started
- Due Sept 13, but pretty open to extensions.



Exercises for Module 1: Intro to Python

This set of exercises works through some basic python functionality. Just a note that we have used some functions from a module called numpy to create the exercises, but nothing you write should need it although you can use it if you'd like. We'll be learning more about numpy in the next module.

```
import numpy as np
```

1.) Write code to translate a boolean value to a string. Specifically, if the `testval` is `True` then print "Yes" and if it is `False` then print "No"

```
testval = bool(np.random.choice([0, 1]))
# TODO: Your code here
message = None
# ...
print(message)
```

2.) You will be given a random integer, and your goal is to return the same value, but as a negative number. Notice, the number you are given may already be negative

```
testval = np.random.random_integers(-100, 100)
# TODO: Your code here
negval = None
print(testval, negval)
```

3.) Given a list of random integers, return them sorted from low to high.

NOTE: I do not want you to write your own sorting algorithm, but want you to look up how to do this using the python standard library

```
random_vals = np.random.random_integers(-100, 100, 10)
# TODO: Your code here
sorted_vals = None
print(sorted_vals)
```

4.) Given a list of US locations with the format: "CityName, StateAbbrev" filter out any that are not in Arizona (AZ).

```
city_list = [
    "New York, NY",
    "Chattanooga, TN",
    "Hobart, MN",
    "Kingman, AZ",
    "Yachats, OR",
    "Bisbee, AZ",
    "Muskogee, OK"
]
#TODO: Your code here
```

Functions and modules

- You have already seen functions - remember the “`type`” command?
  - Same with `str` and `float` and `print`

Functions and modules

- You have already seen functions - remember the “`type`” command?
  - Same with `str` and `float` and `print`
- These have all been “built in” functions, which are provided by python directly.

Functions and modules

- You have already seen functions - remember the “`type`” command?
  - Same with `str` and `float` and `print`
- These have all been “built in” functions, which are provided by python directly.
- Functions provided by the “standard” library give you tons of tools

Functions and modules

- You have already seen functions - remember the “`type`” command?
  - Same with `str` and `float` and `print`
- These have all been “built in” functions, which are provided by python directly.
- Functions provided by the “standard” library give you tons of tools
- But you don’t have direct access to all of them by default
  - They are stored in “modules” which need to be “imported”
  - For example, if you want more math functions you can `import math`

Functions and modules

- You have already seen functions - remember the “`type`” command?
  - Same with `str` and `float` and `print`
- These have all been “built in” functions, which are provided by python directly.
- Functions provided by the “standard” library give you tons of tools
- But you don’t have direct access to all of them by default
  - They are stored in “modules” which need to be “imported”
  - For example, if you want more math functions you can `import math`
- Also, the import system means we can use other people’s code, making our lives vastly easier. We will be learning many important packages in the coming weeks.
-

Functions and modules

- You have already seen functions - remember the "`type`" command?
  - Same with `str` and `float` and `print`
- These have all been "built in" functions, which are provided by python directly.
- Functions provided by the "standard" library give you tons of tools
- But you don't have direct access to all of them by default
  - They are stored in "modules" which need to be "imported"
  - For example, if you want more math functions you can `import math`
- Also, the import system means we can use other people's code, making our lives vastly easier. We will be learning many important packages in the coming weeks.
- But first, let's see how you can write your own functions.

Anatomy of a function

- `def` is the keyword that declares that you want to “define” a new function

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- `def` is the keyword that declares that you want to “define” a new function
- Then, you can give the function a name (note: no spaces allowed)

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- `def` is the keyword that declares that you want to “define” a new function
- Then, you can give the function a name (note: no spaces allowed)
- In parentheses you can state any inputs to the function (empty means no inputs).
- Also, don’t forget the `:`

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- `def` is the keyword that declares that you want to “define” a new function
- Then, you can give the function a name (note: no spaces allowed)
- In parentheses you can state any inputs to the function (empty means no inputs).
- Also, don’t forget the `:`

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- Everything indented from that point on is the “body” of the function

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- Everything indented from that point on is the “body” of the function
- From here you can make use of the input variables to do stuff.

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- Everything indented from that point on is the “body” of the function
- From here you can make use of the input variables to do stuff.
- Finally, when you are done, you can “return” your work.

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- Everything indented from that point on is the “body” of the function
- From here you can make use of the input variables to do stuff.
- Finally, when you are done, you can “return” your work.
- If left out this will default to `None`

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

Anatomy of a function

- Everything indented from that point on is the “body” of the function
- From here you can make use of the input variables to do stuff.
- Finally, when you are done, you can “return” your work.
- If left out this will default to `None`
- A quick note on “scope”

```
# def starts a new function
# |
# |  this is the name of the new function
# |  |
# |  |          this is an input variable to the function
# |  |          |
# v  v          v
def my_function(in_var):
    intermediate = in_var * 8
    out_var = intermediate + 1
    return out_var
#          ^
#          |
#          this is what will come out
```

VSCode interactive session – see recording for more