Python Tools for Machine Learning

Python Tools Useful for Machine Learning

- Simple data types / loops (already covered in form 3 computer literacy)
- Functions
- Modules / libraries
- Classes
- File handling
- Random numbers (also covered in F3 CL, but one crucial part will be mentioned here)
- Exception handling

Functions

- A function takes in arguments (inputs or parameters) and return outputs
- For example, a function that returns the square of a number:

```
def difference(x, y): # define a function with function name and order of arguments
    return x - y # return a value that a function computes
number = difference(2, 3) # call function by function name, inputting arguments IN ORDER
# the variable is assigned to the RETURN VALUE of the function
print(number) # 2 - 3 = -1
```

```
-1
```

Modules

- Sometimes, you need to import things from the external libraries
- This includes functions or other data structures

Modules

```
[5] import math # IMPORTS a module that contains the function / object you want
    print(math.sqrt) # math.sqrt is a FUNCTION
```

```
<built-in function sqrt>
```

```
[6] print(math.sqrt(9)) # call the function
```

```
3.0
```

Think about: why is the return value 3.0 and not 3?

Hint: it’s about data types

Try It Yourself

- Open the Kaggle link https://www.kaggle.com/code/lailaicoding/python-tools with your personal Google account
- Complete the quadratic_equation_solver function

Hint: for $ax^2 + bx + c = 0$,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{19}$$

If $b^2 - 4ac < 0$, there is no real solution; if $b^2 - 4ac = 0$, there is one distinct real solution; if $b^2 - 4ac > 0$, there are two distinct real solutions.

Classes

- A class is a container that makes it convenient to retrieve different objects
- An instance of a class can store variables and functions

Classes

[2]
✓ 0s

```python
class MyClass:
    def __init__(self): # INITIALIZES a class with a SPECIAL KEYWORD: self
        self.b = 5 # once an instance of the class has been created, it can be retrieved with {class name}.b
        self.c = 6 # can be retrieved with {class name}.{variable name}
        # remember to add "self" though for it to be callable from this format
    def add(self): # class instances can also contain functions
        return self.b + self.c # can be called with {class name}.{function name}
    def add_arguments(self, x, y): # self is an argument that is ALWAYS PRESENT in functions contained in classes
        return x + y
a = MyClass()
s = a.add_arguments(2, 4) # the self argument is hidden, so just call the function from the second argument on
print(a.b, a.c, a.add(), s)
```

```
5 6 11 6
```

File Handling

```
with open("file.txt", "w") as wf: # write a file or create one if it doesn't exist
    wf.write("Hello world!") # writes the string argument of the write function to the file
    # you can think of wf as an instance of a class that contains the "write" function
# the file is automatically closed when your code exits the "with open" block
with open("file.txt", "r") as rf:
    contents = rf.read() # read function extracts contents of the file
    print(contents)
# again, the file is closed after the "with open" block, but the variable is saved
print(contents)
```

```
Hello world!
Hello world!
```

File Handling

```python
# sometimes you want to LIST all relevant files in a directory and then store them in a list to loop through them
# in this case the glob.glob function helps
import glob
for item in range(10): # this for loop runs 10 times
    with open(f"file_{item}.txt", "w") as wf: # write a file or create one if it doesn't exist
        wf.write("Hello world!")
    # 10 files are written, file_0.txt through file_9.txt
print(glob.glob("file*.txt"))
```

```
['file_5.txt', 'file_9.txt', 'file_2.txt', 'file_1.txt', 'file_3.txt', 'file_0.txt', 'file_8.txt', 'file_7.txt', 'file.txt', 'file_6.txt', 'file_4.txt']
```

Random Numbers

- Random number generators are good for machine learning purposes, but sometimes you want to replicate that “random” sequence (for experimental tracking purposes for example)
- In this case a random seed can be used: off of the random seed, the “pseudo-random number generator” generates a deterministic sequence of random-looking numbers until a new seed is set

Random Numbers

```python
import random
random.seed(42) # sets the "random seed"
def gen_random_seq():
    lst = []
    for item in range(20):
        lst.append(random.randint(1, 20))
    return lst
print(gen_random_seq()) # first sequence generated with seed 42
print(gen_random_seq()) # second sequence with seed 42, you can see it's different from the first
random.seed(42)
print(gen_random_seq()) # after resetting the seed to the same value, you get the same sequence as the first
```

```
[4, 1, 9, 8, 8, 5, 4, 18, 3, 19, 14, 2, 1, 3, 7, 8, 17, 20, 1, 18]
[7, 18, 14, 8, 15, 19, 9, 1, 6, 14, 11, 9, 5, 7, 11, 4, 3, 13, 4, 12]
[4, 1, 9, 8, 8, 5, 4, 18, 3, 19, 14, 2, 1, 3, 7, 8, 17, 20, 1, 18]
```

Exception Handling

- Maybe your program would throw an error for some reason, and you want to handle it gracefully

```python
def random_generate_error():
    try: # "try" to do something to see if an exception (error) occurs
        if random.random() < 0.5:
            raise ValueError(1)
        print(2)
    except ValueError as e: # "except" statement catches the error
        # the "e" variable name here captures the error message thrown by the "try" block
        print(f"Value Error: {e}")
random.seed(42)
for item in range(5):
    random_generate_error()
# program doesn't terminate on error, instead it gracefully handles it and prints the message, as instructed in the "except" block
```

```
2
Value Error: 1
Value Error: 1
Value Error: 1
2
```

Thank You