Python Basics

February 5, 2020

1 Introduction

Python is a programming language which supports OOP, functional programming, and imperative programming. It is strongly dynamically typed. Strongly typed meaning that variables don't automatically cast to the correct type behind the scenes. Dynamic typing means that variables don't have types, but runtime values (objects) do. So, variables don't require type annotation (things are checked at runtime). If you aren't familiar with any of this vocabulary, don't worry, it is not required to understand this for the assignment.

2 Syntax

Python does not use curly braces '' or semicolons ';' for defining code blocks, but instead uses whitespace. So, it is important to have proper indentation (4 spaces is 1 level of indentation) throughout your code. Code on the same indentation level is part of the same code block.

3 Variables

Variables Python variables do not need to be declared. They are created the first time they are assigned, and exist only within the scope they are created. For example:

```
x = 3
print(x)
```

will print 3, but

```
x = 3
def f():
x = 5
print(x)
f()
print(x)
```

will print

5
3

In python, variables can be ints (integers 1,2,3...), floats (real numbers 1.1,1.32874,5.2398...), 'strings (text, '"hello world"'), bools (boolean values, 'True','False'), lists (like an array or linked list, '[1,2,'hello',3.3982]'), objects (instances of classes), amongst others. You should be familiar with the above types.

Functions In python, you define functions as follows:

```
def example_function(x,y,z):
  return (x+y)/z
```

This function takes in 3 variables, and attempts to do the math as shown. 'def' is the statement which signals you are defining a function, and example_function is the name of the function. Everything below on the indentation level 1 in is part of the function. The 'return' statement exits the function at that line, returning whatever is given in that 'return' statement.

Classes and Objects In python a class is a collection of functions and variables which are related. The class can be initialized to create an object, which can be stored in a variable. You can define a class as follows:

```
class MyClass:
  def __init__(self, x):
    self.x = x
  def printx(self):
    print(self.x)
  def printxplus(self, y):
    print(self.x + y)
  def setx(self, x):
    self.x = x
```

'MyClass' is the name of the class. __init__(self,x) is the initialization function of the class. Note that every class method has the first argument 'self'. When you call a class method on an object, you ignore this argument, as python automatically passes it to the class. As you can see, it is a reference to the object the method is attached to, and allows you to access stored variables. So, to use 'MyClass', you could run some code like:

```
my_class = MyCLass(5)
my_class.printx()
# 5
my_class.printxplus(3)
# 8
my_class.setx(2)
my_class.printx()
# 2
```

The comments show the output of the interpreter at each line. As you can

see, you first initialize an object out of the class. Then, when you call methods, you ignore the 'self' parameter.

Importing Any python file can be imported into another file, giving you access to the classes, function, and variables in the imported file. So, if you have a file called 'lib.py' with a function 'print5' inside of it, you could access it in another file in the same directory with:

```
import lib
lib.print5()
```

You can also import specific things from the file, so you could alternatively do

```
from lib import print5
print5()
```

You will use this so you don't need to rewrite your neural network implementation between parts.

Lists/Arrays In python, lists are defined with the '[]' characters. You can put any python value into a list, including other lists. For example, creating a list with 5 numbers can be done as:

```
l = [1,3,5,7,2]
```

You can index into the list with the '[]' characters as follows:

```
print(l[3])
# 7
```

Note that lists in python are 0 indexed. You can append to lists with the '.append(x)' function:

```
l.append('hello')
print(l)
# [1,3,5,7,2,'hello']
```

To iterate over a list, you can use a for-loop:

```
for x in l:
  print(x)
# 1
# 3
# 5
# 7
# 2
# hello
```

If you also need the indices, you can use the 'enumerate' function.

```
for i,x in enumerate(l):
  print(i,x)
# 0 1
# 1 3
```

```
# 2 5
# 3 7
# 4 2
# 5 hello
```

Finally, if you need to quickly create a list with just numbers, you can use the ‘range‘ function:

```
l = list(range(10))
print(l)
# [0,1,2,3,4,5,6,7,8,9]
```