EE559 Python Instruction Session II: Basics of Python

Part of this tutorial was originally created by Justin Johnson for cs231n. This version has been adapted and improved by Shuai Xu for USC EE559 Spring 2022. It runs Python3 by default. (modified from Min Zhang EE559 S21)

In this tutorial:

- Part 1, we will cover basics of python: **variables and data types, containers, control flow, input/output, function, reading from and writing to files**.
- Part 2, we will cover three important python libraries: **Numpy, Matplotlib, and Pandas**.

A Brief Note on Python Versions

As of Janurary 1, 2020, Python has officially dropped support for `python2`. We'll be using Python 3.7 for this iteration of the course. You can check your Python version in the shell by running `import sys`, then `print(sys.version)`, or at the command line by running `python --version`.

```
import sys
print(sys.version)
```

```
    3.7.12 (default, Jan 15 2022, 18:48:18)
    [GCC 7.5.0]
```

Basics of Python

Python is a high-level, dynamically typed multiparadigm programming language. Python code is often said to be almost like *pseudocode*, since it allows you to express very powerful ideas in very few lines of code while being very readable.

Variables & Data Types

Variable is basically just a container where we can store certain data values, it makes a lot easier for us to work with and manage all of the different data inside of our programs. We will look at the different types of variables and the different types of data that we can store inside of variables.

```
print("There is a course called EE559 in USC, ")
print("one of the TAs is Thanos.")
```

```
    There is a course called EE559 in USC,
    one of the TAs is Thanos.
```

In order to make that change, you will need to look through the entire program find it where that value was and change it.

```
print("There is a course called EE660 in USC, ")
print("one of the TAs is Fernando.")
```

```
    There is a course called EE660 in USC,
    one of the TAs is Fernando.
```

```
course_name = "EE660"  # only string needs quotation mark
ta_name = "Fernando"
print("There is a course called " + course_name + " in USC, ")
print("one of the TAs is " + ta_name + ".")
```

```
    There is a course called EE660 in USC,
    one of the TAs is Fernando.
```

Strings

```
# String literals can use single quotes or double quotes in Python; it does not mat
print('hello\nworld')  # insert a new line into the string
```

```
    hello
    world
```

```
print('Shuai\'s Apple')  # put a quotation mark into the string using a escape char
```

```
Shuai's Apple
```

```
hello = 'hello'
world = "world"
hw = hello + ' ' + world  # String concatenation
print(hw)
```

```
hello world
```

```
hw12 = '{} {} {}'.format(hello, world, 2022)  # string formatting
print(hw12)
hw12 = f'{hello} {world} {2022}'
print(hw12)
```

```
hello world 2022
hello world 2022
```

String objects have a bunch of useful methods/functions. You can find a list of all string methods in the documentation. For example:

```
s = "hello"
print(s.capitalize())           # Capitalize a string
print(s.upper())                # Convert a string to uppercase; prints "HELLO"
print(s.isupper())              # Check if a string is entirely uppercase
print(s.upper().isupper())
print(s.rjust(7))               # Right-justify a string, padding with spaces
print(s.center(7))              # Center a string, padding with spaces
print(s.replace('l', '(ll)'))  # Replace all instances of one substring with anothe
print('  world '.strip())       # Strip leading and trailing whitespace
```

```
Hello
HELLO
False
True
  hello
 hello
he(ll)(ll)o
world
```

```
s = "hello"
print(len(s))          # length of the string, really useful
print(s[0])            # indexing starts at 0
print(s[:2])           # slicing
print(s.index("ll"))
# print(s.index("z"))  # raise error if the character doesn't exist
```

```
5
h
he
2
```

```
s[0] = 'T'             # immutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-5d1343035819> in <module>()
----> 1 s[0] = 'T'             # immutable

TypeError: 'str' object does not support item assignment
```

SEARCH STACK OVERFLOW

Numbers

Integers and floats work as you would expect from other languages:

```
x = 3.1
print(x, type(x))
```

```
3.1 <class 'float'>
```

```
print(x + 1)   # Addition
print(x - 1)   # Subtraction
print(x * 2)   # Multiplication
print(x ** 2)  # Exponentiation
print(x % 2)   # Remainder
```

```
4.1
2.1
6.2
9.610000000000001
1.1
```

```
x += 1   # x = x + 1, can not use x++
print(x)
x *= 2  # x = x * 2
print(x)
```

```
4.1
8.2
```

```
y = 2.5
print(type(y))
print(y, y + 1, y * 2, y ** 2)
```

```
<class 'float'>
2.5 3.5 5.0 6.25
```

Note that unlike many languages, Python does not have unary increment (x++) or decrement (x--) operators.

Python also has built-in types for long integers and complex numbers; you can find all of the details in the documentation.

```
z = 3
print(str(z) + " my favorite number")  # print number next to a string
print(z + " my favorite number")     # raise error
```

```
3 my favorite number
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-dade119c81f5> in <module>()
      1 z = 3
      2 print(str(z) + " my favorite number")  # print number next to a
string
----> 3 print(z + " my favorite number")     # raise error

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
z = -5
print(abs(z))
print(pow(z, 3))  # **
print(max(4, 6))
print(round(3.8))
print(floor(3.7))
```

```
5
-125
6
4
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-51b24397a951> in <module>()
      4 print(max(4, 6))
      5 print(round(3.8))
----> 6 print(floor(3.7))

NameError: name 'floor' is not defined
```

SEARCH STACK OVERFLOW

To access some other functions, we should import external code into our files.

```
import math  # from math import *
z = 3.7
print(math.floor(z))     # grab the lowest integer
print(math.ceil(z))
print(math.sqrt(4))
```

```
3
4
2.0
```

Booleans

Python implements all of the usual operators for Boolean logic

```
t, f = True, False
print(type(t))
```

```
<class 'bool'>
```

```
print(t and f) # Logical operator AND;
print(t or f)  # Logical operator OR;
print(not t)   # Logical operator NOT;
print(t & f)   # symbol &
print(t | f)   # symbol |
print(t != f)  # Logical XOR; XOR: if not equal, we get True
```

```
False
True
False
False
True
True
```

```
c = input("What's your name: ")
print(type(c))
print("My name is " + c)
```

```
What's your name: Shuai
<class 'str'>
My name is Shuai
```

Containers

Python includes several built-in container types: lists, dictionaries, sets, and tuples.

Lists

A list is the Python equivalent of an array, but is resizeable and can contain elements of different types:

```
xs = [3, 1, 2]    # Create a list
print(xs)
print(xs[2])
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
```

```
[3, 1, 2]
2
2
```

```
xs[2] = 'EE559'    # Lists can contain elements of different types
print(xs)
```

```
[3, 1, 'EE559']
```

```
xs.append('bar') # Add a new element to the end of the list
print(xs)
```

```
[3, 1, 'EE559', 'bar']
```

```
x = xs.pop()     # Remove and return the last element of the list
print(x, xs)
```

```
bar [3, 1, 'EE559']
```

As usual, you can find all the gory details about lists in the documentation.

Slicing

In addition to accessing list elements one at a time, Python provides concise syntax to access sublists; this is known as slicing:

```
nums = list(range(5))    # range is a built-in function that creates a list of inte
print(nums)              # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])         # Get a slice from index 2 to 4 (exclusive); prints "[2, 3
print(nums[2:])          # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])          # Get a slice from the start to index 2 (exclusive); print
print(nums[:])           # Get a slice of the whole list; prints ["0, 1, 2, 3, 4]"
print(nums[:-1])         # Slice indices can be negative; prints ["0, 1, 2, 3]"
print(nums[1:4:2])       # step = 2, [1, 9]
print(nums[::-1])        # step = -1, reverse
nums[2:4] = [8, 9, 10, 10]       # Assign a new sublist to a slice
print(nums)              # Prints "[0, 1, 8, 9, 4]"
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[1, 3]
[4, 3, 2, 1, 0]
[0, 1, 8, 9, 10, 10, 4]
```

For Loop

You can loop over the elements of a list like this:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

List comprehensions:

When programming, frequently we want to transform one type of data into another. As a simple example, consider the following code that computes square numbers:

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)
```

```
[0, 1, 4, 9, 16]
```

You can make this code simpler using a list comprehension:

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)
```

```
[0, 1, 4, 9, 16]
```

List comprehensions can also contain conditions:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)
```

```
[0, 4, 16]
```

Dictionaries

A dictionary stores (key, value) pairs, similar to a `Map` in Java or an object in Javascript. You can use it like this:

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
```

```
cute
True
```

```
d['fish'] = 'wet'    # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
print(d['monkey'])  # KeyError: 'monkey' not a key of d
```

```
wet
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-33-2101ff471e2e> in <module>()
      1 d['fish'] = 'wet'    # Set an entry in a dictionary
      2 print(d['fish'])      # Prints "wet"
----> 3 print(d['monkey'])  # KeyError: 'monkey' not a key of d

KeyError: 'monkey'
```

SEARCH STACK OVERFLOW

```
print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
```

```
N/A
wet
```

```
del d['fish']        # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

```
N/A
```

You can find all you need to know about dictionaries in the documentation.

It is easy to iterate over the keys in a dictionary:

```
d = {'monkey': 2, 'cat': 4, 'spider': 8}
print(d.items())
```

```
dict_items([('monkey', 2), ('cat', 4), ('spider', 8)])
```

```
for animal, legs in d.items():
    print('A {} has {} legs'.format(animal, legs))
```

```
A monkey has 2 legs
A cat has 4 legs
A spider has 8 legs
```

Dictionary comprehensions: These are similar to list comprehensions, but allow you to easily construct dictionaries. For example:

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)
```

```
{0: 0, 2: 4, 4: 16}
```

Sets

A set is an **unordered** collection of **distinct** elements. As a simple example, consider the following:

```
new = {3, 1.2, True, (1, 2)}  # set can't contain list
print(new)
animals = {'cat', 'dog'} # set keeps distinct elements
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False"
```

```
{(1, 2), 1.2, 3, True}
True
False
```

```
animals.add('fish')      # Add an element to a set, unordered
print(animals)
print('fish' in animals)
print(len(animals))       # Number of elements in a set;
```

```
{'dog', 'cat', 'fish'}
True
3
```

```
animals.add('cat')       # Adding an element that is already in the set does nothir
print(animals)
animals.remove('cat')    # Remove an element from a set
print(animals)
```

```
{'dog', 'cat', 'fish'}
{'dog', 'fish'}
```

Set comprehensions: Like lists and dictionaries, we can easily construct sets using set comprehensions:

```
from math import sqrt
print({int(sqrt(x)) for x in range(5)})
```

```
{0, 1, 2}
```

Tuples

A tuple is an **immutable** ordered list of values. A tuple is in many ways similar to a list; one of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot. Here is a trivial example:

```
t = (5, 6)       # Create a tuple
print(type(t))
d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys, but
print(d[t])
print(d[(1, 2)])
```

```
<class 'tuple'>
5
1
```

```
print(t[0])
t[0] = '1'   # immutable
```

```
5
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-0d32a1efced0> in <module>()
      1 print(t[0])
----> 2 t[0] = '1'   # immutable

TypeError: 'tuple' object does not support item assignment
```

SEARCH STACK OVERFLOW

```
coordinates = [(1, 2), (3, 4)]   # list of tuples
print(coordinates[0])
coordinates[0] = 1
print(coordinates)
```

```
(1, 2)
[1, (3, 4)]
```

Control Flow

If Statement

Execute certain code when certain conditions are true.

```
a = 0
if a > 0:
  print("a is positive")
elif a == 0:
  print("a is zero")
else:
  print("a is negative")
```

```
a is zero
```

For Loop

You can loop over the elements of a list like this:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

If you want access to the index of each element within the body of a loop, use the built-in `enumerate` function:

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#{}: {}'.format(idx + 1, animal))
```

```
#1: cat
#2: dog
#3: monkey
```

While Loop

Loop through the code executing it repeatedly until a certain condition was false.

```
i = 1
while i <= 10:
  print(i)
  i += 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Try Except

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called *exceptions* and are not necessarily fatal: exceptions can be caught and dealt with within the program so that our program won't break.

```
number = int(input("Enter a number: "))
print(number)
print("continue")
```

```
Enter a number: Shuai
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-50-d76eec5e7be6> in <module>()
----> 1 number = int(input("Enter a number: "))
      2 print(number)
      3 print("continue")

ValueError: invalid literal for int() with base 10: 'Shuai'
```

SEARCH STACK OVERFLOW

```
try:
  number = int(input("Enter a number: "))
  print(number)
except:
  print("Invalid Input")
print("continue")
```

```
Enter a number: Shuai
Invalid Input
continue
```

```
value = 10 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-52-6636284eb0ed> in <module>()
----> 1 value = 10 / 0

ZeroDivisionError: division by zero
```

SEARCH STACK OVERFLOW

```
try:
  value = 10 / 0
except ZeroDivisionError:  # catch different types of errors
  print("Divided by zero")
except ValueError:
  print("Invalid input")
```

```
    Divided by zero
```

```
try:
  value = 10 / 0
except ZeroDivisionError as err:  # store as variable
  print(err)
except ValueError:
  print("Invalid input")
```

```
    division by zero
```

Functions

Function is basically just a collection of code which performs a specific task, helping organize your code.

Python functions are defined using the `def` keyword.

```
def signNewElement(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(signNewElement(x))
```

```
    negative
    zero
    positive
```

We will often define functions to take optional keyword arguments, like this:

```
def hello(name, loud=False):
    if loud:
        print('HELLO, {}'.format(name.upper()))
    else:
        print('Hello, {}!'.format(name))

hello('Bob')
hello('Fred', loud=True)
```

```
Hello, Bob!
HELLO, FRED
```

Reading and Writing Files

1. Write some text to a file with name test.txt
2. Read the text again
3. Print it to the screen

```
f = open("test.txt", "w")  # w: write
f.write("This is the first line.\n"
        "This is the second line.\n"
        "This is the last line.")
f.close()
```

```
f = open("test.txt", "a")   # a: append
f.write("\nThis is the new line.")
f.close()
```

```
f = open("test.txt", "w")
f.write("\nThis is the new line.")  # if "r", can't write
print(f.readable())
f.close()
```

```
False
```

```
f = open("test.txt", "r")   # r: read, r+: read and write
print(f.readable())
print(f.read())
f.close()
```

```
True

This is the new line.
```

```
f = open("test.txt", "r")
print(f.readline())   # read first line
# print(f.readline())  # read second line
f.close()
```

```
f = open("test.txt", "r")
print(f.readlines())
f.close()
```

```
['\n', 'This is the new line.']
```

```
# no need to close the file by yourself
with open("test.txt", "r") as f:
  for line in f.readlines():
    print(line)
```

```

This is the new line.
```

Example

As an example, here is an implementation of the classic quicksort algorithm in Python

```
def quicksort(arr):                                    # function with required arg
    if len(arr) <= 1:                                  # if statement
        return arr
    pivot = arr[len(arr) // 2]                         # list indexing
    left = [x for x in arr if x < pivot]               # list comprehension with co
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)  # + is append between lists

print(quicksort([3,6,8,10,1,2,1]))                      # input is a list of number
```

```
[1, 1, 2, 3, 6, 8, 10]
```

———— Next Time ————

Numpy, Matplotlib, Pandas

Numpy

[ ] ↳ *82 cells hidden*

Matplotlib

[ ] ↳ *16 cells hidden*

Pandas

[ ] ↳ *62 cells hidden*

———— Next Time ————

Object oriented programming in Python.