A crash course in Python
Written by Dr. K. Kalpakis
The material in this notebook is modeled after Chapter 2 (A Crash Course in Python) in the book by Grus

0.1 Start by getting some libraries first

In [1]: # Importing a module (package/library) into python
    import sys, os

In [2]: #some common imports
    from __future__ import division
    import re
    import math

    #use new handle (alias) for imported library
    import matplotlib.pyplot as plt

0.1.1 Calling functions from imported modules

In [3]: math.sqrt(20) # call the sqrt function from the module math

Out[3]: 4.47213595499958

0.2 Strings

Define strings using either ‘single’ or "double" quotes; use triple quotes used for multiline strings
Use ” to escape special characters.

In [4]: s1 = "First String"
   s2 = 'second String'
   multiLine_string = """This is a very long multi Line String. Which can go on, and on, and on...""
   tab = "\t"
   notab1 = "\\t"
   notab2 = r'\t' # raw string
In [5]:
    print(s1)
    print(s2)
    print(multiLine_string)
    print(tab)
    print(notab1)
    print(notab2)

First String
second String
This is a very long
multi Line String. Which can go on, and on, and on...

\t
\t

0.2.1 Indentation using tabs or whitespaces to define the scope of blocks of code

In [6]: # Print numbers from 2 up to 20 with stride of 7
    for i in range(2,20,7):
        # body of the loop is indented
        j = i * math.log(i)
        print(i,j)
    # end of the loop body

2 1.3862943611198906
9 19.775021196025975
16 44.3614195558365

0.3 Control Flow

0.3.1 if.. elif.. else

In [7]: score = 92
    if (score >= 90):
        grade = 'A'
    elif (80 <= score < 90):
        grade = 'B'
    elif (70<= score < 80):
        grade = 'C'
    else:
        grade = 'F'
    print(score, grade)

92 A
0.3.2 while loop

In [8]: x = 0
    while (x < 4):
        print (x, math.exp(x))
        x += 1

0 1.0
1 2.718281828459045
2 7.38905609893065
3 20.085536923187668

0.3.3 for loop

In [9]: for x in range(5):
    print (x)

0
1
2
3
4

0.3.4 Indentation in case of nested for loops

In [10]: # An example of nested for loops
    for i in range(2):
        for j in range(2,5):
            print(i,j, i+j)

0 2 2
0 3 3
0 4 4
1 2 3
1 3 4
1 4 5

0.4 Functions

In [11]: # define a function with 2 arguments and default values for its parameters
    def addition(arg1=100, arg2=200):
        # Body of the function
        t = arg1 + arg2
        return t

In [12]: # ways to call a function with actual parameter values
# match parameter values to function arguments by position
print(addition(5, 10))

print(addition()) # use only the defaults

# provide only named parameter value, rest take their defaults
print(addition(arg2=20))

# call with parameters given out of order
print(addition(arg2=20, arg1=3))

15
300
120
23

In [13]: # function call without parentheses
addition

Out[13]: <function __main__.addition>

0.4.1 Assign functions to variables

In [14]: # Assign the function to another variable
myAdd = addition
myAdd(20) # call function reference

Out[14]: 220

In [15]: # Dynamically typed
myAdd("Data ", "Science")

Out[15]: 'Data Science'

0.4.2 Pass functions as arguments to functions

In [16]: # Create a new function for fancy printing
def prettyPrint(data):
    print('\\t', data(), '\\n')

In [17]: # Pass a function as an argument to it
prettyPrint(myAdd)

[ 300 ]
0.5 Exceptions

In [18]: try:
    ...:     print (0/0)
    ...:     except ZeroDivisionError:
    ...:         print ("division by 0 exception")

division by 0 exception

0.6 Lists

An ordered collection of things of various types/kinds. Core Python data structure.

In [19]: list1 = [10, 20, 30, 40, 50]
    
    list2 = ["Apple", 5.0, 42]
    
    list_of_lists = [[100,200,300], list1]

In [20]: list1
Out[20]: [10, 20, 30, 40, 50]

In [21]: list_of_lists
Out[21]: [[100, 200, 300], [10, 20, 30, 40, 50]]

0.6.1 lists are 0-indexed (similar to arrays in C++)

In [22]: # List elements can be accessed with index.
    ...:     list1[4]

Out[22]: 50

In [23]: # Pass lists as arguments (+ concatenates the two operand lists)
    ...:     addition(list1, list2)

Out[23]: [10, 20, 30, 40, 50, 'Apple', 5.0, 42]

0.6.2 Slicing lists

In [24]: # sublist of first 3 elements
    ...:     list1[:3]

Out[24]: [10, 20, 30]

In [25]: # another sublist using strides (first location:last location:stride)
    ...:     list1[1:4:2]

Out[25]: [20, 40]

In [26]: # the 2nd element from the end of the list
    ...:     list1[-2]

Out[26]: 40
0.6.3 list membership

In [27]: for i in range(0,100,10):
    ...:     if (i in list1):
    ...:         print(i, "Yes")
    ...:     else:
    ...:         print(i, "No")

0 No
10 Yes
20 Yes
30 Yes
40 Yes
50 Yes
60 No
70 No
80 No
90 No

In [28]: # append/insert/search/remove elements from/to a list
   ...: list1 = [10, 20, 30, 40, 50]
   ...:   
   ...: # append element at the end of a list
   ...: print(list1.append(60), list1)
   
   ...: #remove last list element, and return that element
   ...: print(list1.pop(), list1)
   
   ...: #remove element at given position
   ...: print(list1.pop(2), list1)
   
   ...: # find the position of given value in list
   ...: print(list1.index(40), list1)
   
   ...: # insert a value 30 at position 2 of list
   ...: print(list1.insert(2, 30), list1)

None [10, 20, 30, 40, 50, 60]
60 [10, 20, 30, 40, 50]
30 [10, 20, 40, 50]
2 [10, 20, 40, 50]
None [10, 20, 30, 40, 50]

0.6.4 Python trick: swap variables using lists

In [29]: a = 5
   ...: b = 10
print(a,b)
a, b = b, a  # swap (assignment of elements across two lists)
print(a,b)

5 10
10 5

0.7 Tuples

Immutable lists. All list operations that do not modify the list are supported on tuples Use (...) instead of [...]

In [30]: x = (1,2,3)
In [31]: # Tuple elements can be accessed with index.
   x[2]
Out[31]: 3

0.7.1 Let's try changing the value of the tuple at an index

In [32]: # Tuples are immutable
   try:
      x[1] = 10
   except TypeError:
      print('error :: tuples are immutable')
error :: tuples are immutable

In [33]: # dynamic selection of tuple element
   i = 5
   val = ("Even", "Odd")[i % 2 != 0]
   print(val)
Odd

0.8 Sets

A collection of distinct things (faster membership tests for sets than lists)

In [34]: s = set()
s.add(1) # s is now { 1 }
s.add(2) # s is now { 1, 2 }
s.add(2) # s is still { 1, 2 }

   x = len(s) # length of set
   y = 2 in s # set membership
   z = 3 in s
   print(s, x, y, z)
In [35]:
dups = [0,1,2,2,1,0,0,1]  # list with duplicates
noDups = set(dups)  # set - no duplicates
print(noDups)

{0, 1, 2, 3}

In [36]:  # intersect two sets
    set(set([0,1,2,3]) & set([2,5,10]))

Out[36]: {2}

0.9 Dictionaries
Mutable collections for storing (key : value) pairs

In [37]:  # Dictionary of names and ages
    name_age_dict = {
        "Alex" : 20,
        "Amanda" : 22,
        "Ruben" : 43,
        "Dave" : 6
    }

    # Dict of names and courses
    name_course_dict = {
        "Alex" : ["404", "461", "469"],
        "Amanda" : ["321", "345", "404"],
        "Ruben" : ["691", "622"],
        "Dave" : []
    }

In [38]:  # Accessing key:value pairs in a dict
    for key, value in name_age_dict.items():
        print(key, ":", value)

        Alex : 20
Amanda : 22
Ruben : 43
Dave : 6

In [39]:  # Adding/modifying value of a key in dictionary
    name_age_dict["Tom"] = 12  # new key
    name_age_dict["Amanda"] = 21  # update existing key
In [40]: for key, value in name_age_dict.items():
    print(key, ":", value)

Alex : 20
Amanda : 21
Ruben : 43
Dave : 6
Tom : 12

In [41]: # Get value associated with key
    print(name_age_dict["Ruben"])

43

In [42]: # Key error in case of missing keys
    try:
        print(name_age_dict["Yoda"])
    except KeyError:
        print('error :: key not found')

error :: key not found

In [43]: # To avoid KeyErrors use get() method which returns None
    print("Ruben's age is ", name_age_dict.get("Ruben"))
    print("Yoda's age is ", name_age_dict.get("Yoda"))

    # get() with default
    print("Yoda's age is ", name_age_dict.get("Yoda", 150))

Ruben's age is 43
Yoda's age is None
Yoda's age is 150

0.9.1 Default dictionaries
append-to-value of key (pre-insert 'empty' value if key was missing prior); allows to easily implement accumulators/reducers

In [44]: from collections import defaultdict

dd = defaultdict(int)
dd["ls"] += 1
print(dd)

dd = defaultdict(list) # list() produces an empty list
dd["mv"].append([1, 'abc']) # now dd_list contains {2: [1, 'abc']}
print(dd)
defaultdict(<class 'int'>, {'ls': 1})
defaultdict(<class 'list'>, {'mv': [[1, 'abc']]])

0.10 Booleans
possible values are 'True', 'False', 'None' The expressions below evaluate to 'False'

In [45]: x = False or None or [] or {} or '' or set() or 0 or 0.0
    print(x)
    if (x):
        print("True")
    else:
        print("False")

0.0
False

0.10.1 Special functions: 'all' and 'any'
'all' is an aggregated logical 'and', while 'any' is an aggregated logical 'or'

In [46]: vals = [True, 1.0, False]
    print( all(vals) ) #evaluates to False
    print( any(vals) ) #evaluates to True

False
True

0.10.2 Python trick: (emulating) the ternary conditional operator ? of C

In [47]: # Using if..else
    x = True
    x = False
    val = "cat" if x else "dog"
    print(val)

dog

1 Part II

1.1 Build-in sorting functions
'.sort()' method does in-place sorting for lists only sorted() function returns a sorted version of any iterator
In [48]: aList = [8, 4, 5, 12, -7]

    # printing the sorted List
    print(sorted(aList))

    # initial list is still unsorted
    print(aList)

    # in-place sorting of list is changed to its sorted version
    aList.sort()
    print(aList)

[-7, 4, 5, 8, 12]
[8, 4, 5, 12, -7]
[-7, 4, 5, 8, 12]

sorting order is smallest to largest by default; can be changed by using the 'reverse' flag

In [49]: aList.sort(reverse=True)
    print(aList)

[12, 8, 5, 4, -7]

1.2 List comprehensions

Powerful, elegant and efficient way of playing with lists.

In [50]: all_numbers = [x for x in range(5)] # gives [0, 1, 2, 3, 4]
    even_numbers = [x for x in range(5) if x % 2 == 0] # gives [0, 2, 4]
    squares = [x * x for x in all_numbers] # gives [0, 1, 4, 9, 16]
    even_squares = [x * x for x in even_numbers] # gives [0, 4, 16]

    print('All numbers:', all_numbers)
    print("Evens: ", even_numbers)
    print("Squares: ", squares)
    print("Even Squares", even_squares)

All numbers: [0, 1, 2, 3, 4]
Evens: [0, 2, 4]
Squares: [0, 1, 4, 9, 16]
Even Squares [0, 4, 16]

In [51]: # compute list intersection via list comprehensions
    [x for x in squares if x in even_numbers]

Out[51]: [0, 4]
1.2.1 List comprehension works for sets and dictionaries as well.

```
list = [ expression for .... if... ]
```

```
# associate k with string of k bars
dict_comprehension = {x : "|"*x for x in range(5)}
```

```
#make set with elements 1 ... 9
set_comprehension = set(i for i in range(10) if i >0)
```

```
print("\nDict: ", dict_comprehension, "\n\nSet: ", set_comprehension)
```

```
Set: {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

1.2.2 create a 2D matrix using list comprehensions

```
# create a 4x5 matrix of tuples (i,j,val) where val = i+j in a row-major way
matrix = [ [(i,j, i+j) for j in range(5)] for i in range(4) ]
for row in matrix:
    print(row)
```

```
[(0, 0, 0), (0, 1, 1), (0, 2, 2), (0, 3, 3), (0, 4, 4)],
[(1, 0, 1), (1, 1, 2), (1, 2, 3), (1, 3, 4), (1, 4, 5)],
[(2, 0, 2), (2, 1, 3), (2, 2, 4), (2, 3, 5), (2, 4, 6)],
[(3, 0, 3), (3, 1, 4), (3, 2, 5), (3, 3, 6), (3, 4, 7)]
```

```
Out[53]: [[(0, 0, 0), (0, 1, 1), (0, 2, 2), (0, 3, 3), (0, 4, 4)],
[(1, 0, 1), (1, 1, 2), (1, 2, 3), (1, 3, 4), (1, 4, 5)],
[(2, 0, 2), (2, 1, 3), (2, 2, 4), (2, 3, 5), (2, 4, 6)],
[(3, 0, 3), (3, 1, 4), (3, 2, 5), (3, 3, 6), (3, 4, 7)]]
```

1.3 Enumerate

using both the index and value from an iterator

```
#Enumerate: return a list of (index,value) pairs for the values in the input list
objs = ['a','b', ('c', 'd', 2)]

# make list of those (index,value) pairs
pairs = [(i,v) for i,v in enumerate(objs)]
print(pairs)
```

```
[(0, 'a'), (1, 'b'), (2, ('c', 'd', 2))]
```
1.4 Regular expressions

In [55]: # basic regular expression operations
    import re

    # match(E, S) to find a match of regexp E to string S
    print(re.match("a", "cat")) # no matches since 'cat' doesn't match 'a'
    print(re.search('a', 'cat')) # 'cat' contains 'a'
    print(re.search('c', 'dog')) # 'dog' does not have a 'c'

    print(re.split("[ab]", 'carbs')) # split 'carbs' based on 'a' and 'b'

    print(re.sub("[0-9]", "-", "R2D2")) # replace digits with dashes

None
<_sre.SRE_Match object; span=(1, 2), match='a'>
None
['c', 'r', 's']
R-D-

1.5 Randomness

In [56]: import random

    #get some random reals (as a list comprehension)
    X = [random.random() for _ in range(4)] # note the anonymous iterator variable _
    print(X)

    #random element from a range with a stride
    print(random.randrange(3, 20, 3))

    #some random integers between two endpoints
    print([random.randint(2, 20) for _ in range(5)])

    #some random values from a Normal distribution
    print([random.gauss(2, 0.2) for _ in range(5)])

[0.47397949904811254, 0.42561630600711686, 0.16331641908969186, 0.6909498035928557]
15
[16, 19, 5, 19, 13]
[2.0504613074720086, 2.3407403790286145, 1.93411686609859, 2.0182312034112733, 2.1923695052004035]

In [57]: # Randomly choose an item from a list
    suites = ["Diamond", "Spade", "Club", "Heart"]
    v = random.choice(suites)
    print(v)
Spade

In [58]: # Randomly select k elements from a list without replacement
lottery_numbers = range(60)
winning_numbers = random.sample(lottery_numbers, 6)
print(winning_numbers)

[30, 54, 26, 32, 39, 35]

In [59]: # Shuffle in-place a list at random
nums = list(range(10))
random.shuffle(nums)
print("Shuffled: ", nums)

Shuffled: [2, 8, 5, 6, 4, 1, 0, 9, 3, 7]

1.6 Functional programming: map, filter, and reduce

In [60]: # MAP: apply a function over an iterator
# use an anonymous function (lambda function)
nums = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x**2, nums))
print(squares)

[1, 4, 9, 16, 25]

In [61]: # FILTER(func, iter): select those elements x of an iterator that
# have func(x) True. Returns an iterator

# retain the even elements in squares
even_squares = list(filter(lambda x: x % 2 == 0, squares))

print(even_squares)

[4, 16]

In [62]: # REDUCE: Successively use a single operator on all elements
# of an iterator; returns a single value

# import the reduce from the functools module so that it can be used directly
from functools import reduce

sum_of_squares = reduce(lambda x, y: x+y, squares)
print(sum_of_squares)

55
1.7 Zip and Unzip

Zip does argument unpacking. Think of 'the first thing that pops in head': your jacket's zipper. It's exactly that!

In [63]: # Combine corresponding elements (same index) from two input lists into tuples
   
   objsA = ['a','b','c','d','e']
   objsB = [1, 2, 3, (4, 'base')]

   objs = list(zip(objsA, objsB))
   print(objs)

   [('a', 1), ('b', 2), ('c', 3), ('d', (4, 'base'))]

unzip does argument unpacking. The '*' performs argument unpacking. Uses elements of pairs as individual arguments to zip.

In [64]: # Separate out the pairs in objs into two lists
   
   objsA, objsB = zip(*objs)

   # The above call is equivalent to zip(('a',1), ('b',2), ('c',3), ('d',4))
   print(objsA)
   print(objsB)

   ('a', 'b', 'c', 'd')
   (1, 2, 3, (4, 'base'))

1.8 Object-oriented programming

define new classes

In [65]: # Define a new class

   class DS_Set:
       # these are the member methods take a first parameter "self"
       # (a Python convention) referencing the current class instance

       def __init__(self, values=None):
           """This is the class constructor.
           s1 = DS_Set() # empty set
           s2 = DS_Set([1,2,2,3]) # initialize with values"
           self.dict = {} # each instance has its own dictionary to track its members
           if values is not None:
               for v in values:
                   self.add(v)

       def __str__(self):
DS_Set: dict_keys([1, 2, 3, 4])
True
False

1.9 Generators and lazy iterators
Generator: an object that you iterate over whose values are produced as needed (lazily)

In [67]: # define a lazy iterator. Notice the 'yield' instead of 'return'
def lazy_evens(n):
    i = 0
    while i < n:
        yield i
        i += 2
In [68]: %timeit 1
#time the execution of this cell with 20 executions
# use items of the generator lazy_evens as needed
for i in lazy_evens(1000000000000):
print (i)
if i >= 10:
    break

100000000 loops, best of 3: 7.38 ns per loop
0
2
4
6
8
10

In [69]: %timeit -n 1000000 1
   # can also make generator by wrapping a list comprehension in parentheses
large_evens = (i for i in lazy_evens(200000000) if i > 100)
for i in large_evens:
    print(i);
    if i > 120:
        break

1000000 loops, best of 3: 6.76 ns per loop
102
104
106
108
110
112
114
116
118
120
122

1.10 Functions with variable arguments

In [70]: # Define a function with variable (number of) arguments
    def foo_varargs(*args, **kwargs):
        # print out the arguments:
        for a in args:
            print(a)
        for key, value in kwargs.items():
            print(key, value)
        print('Body of the function')
    return None

    # define parameter values and call the function
    args = [1,2,"three"]
kwargs = {'A': 1, 'B': 2, 'C': 3}

foo_varargs(args, kwargs)

[1, 2, 'three']
{'A': 1, 'B': 2, 'C': 3}

Body of the function

1.10.1 Splitting python code into multiple files (modules)

Can put some functions (say foo), classes etc in a file, eg 'my_code.py' Then, in notebook you can import my_code as mine and call then as mine.foo() Alternatively, can import all (*) or select items from my_code import foo from my_code import * Can also create a directory module with multiple files and a '__init__.py' as below

```
my_code/
__init__.py
  #contents of __init__.py are here
  from .partA import classA
  from .partB import fooB
partA.py
  class classA():
partB.py
  def fooB():
```

and then import the module and use as import my_code as mine mine.fooB()

1.11 Including LaTeX markdowns in the notebook

The LaTeX equations are interpreted by the notebook.

\[ e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \]

Any other LaTeX commands are interpreted when the notebook downloaded as .tex or .pdf.

• a \( \frac{2^x}{3+y} \)
• b

1.12 Build-in magic jupyter notebook commands

get info about magic commands
    enter the debug mode for the cell
time the execution of the cell list of global scope (interactive) variables
writes the rest of a cell’s contents into a file
replaces the cell’s contents with the contents of the filename
export and save notebook to filename Print the call signature for any callable object. profile cell execution ?object delete var with given name
In [71]: %latex
   
   Render the contents of the cell as LaTeX, eg
   \[ \sum_i e^{t_i} \]

   Render the contents of the cell as LaTeX, eg \( \sum_i e^{t_i} \)

In [72]: %html
   
   # Render the contents of the cell as HTML
   <table>
   <tr><td>1</td><td>2</td></tr>
   <tr><td>a</td><td>b</td></tr>
   </table>

   <IPython.core.display.HTML object>

1.12.1 Export notebook as slides (HTML slideshow)

Can create a static HTML-based slideshow of the notebook

   jupyter nbconvert mynotebook.ipynb --to slides

   Ensure that you have the reveal.js in the directory of the saved slideshow before opening it with browser

   In addition, a live slideshow of the notebook can be rendered using the RISE Jupyter extension (which creates a button on the notebook’s toolbar).

   See https://damianavila.github.io/RISE/ for installation instructions of RISE to your Jupyter notebook viewer. !pip install RISE !jupyter-nbextension install rise --py --sys-prefix !jupyter-nbextension enable rise --py --sys-prefix