Data Science

Introduction to Python Programming

Formatting

- Many languages use curly braces to delimit blocks of code. **Python uses indentation.** Incorrect indentation causes error.
- Comments start with #
- Colons start a new block in many constructs, e.g. function definitions, if-then clause, for, while

```
for i in [1, 2, 3, 4, 5]:
    # first line in "for i" block
    print (i)
    for j in [1, 2, 3, 4, 5]:
        # first line in "for j" block
        print (j, end=' ') # end=' ' for horizontal print
        # last line in "for j" block
        print (i + j)
    # last line in "for i" block print "done looping
    print (i)
print ("done looping”)
```

Whitespace is ignored inside parentheses and brackets.

```
long_winded_computation = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 +
                9 + 10 + 11 + 12 + 13 + 14 +
                15 + 16 + 17 + 18 + 19 + 20)

list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

easier_to_read_list_of_lists =
[ [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9] ]
```

Alternatively:

```
long_winded_computation = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + \
                9 + 10 + 11 + 12 + 13 + 14 + \
                15 + 16 + 17 + 18 + 19 + 20
```

Modules

- Certain features of Python are not loaded by default
- In order to use these features, you'll need to import the modules that contain them.
- E.g.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Variables and objects

- Variables are created the first time it is assigned a value
  - No need to declare type
  - Types are associated with objects not variables
    - X = 5
    - X = [1, 3, 5]
    - X = ‘python’
  - Assignment creates *references*, not *copies*

```
X = [1, 3, 5]
Y= X
X[0] = 2
Print (Y) # Y is [2, 3, 5]
```

Assignment

- You can assign to multiple names at the same time

```
x, y = 2, 3
```

- To swap values

```
x, y = y, x
```

- Assignments can be chained

```
x = y = z = 3
```

- Accessing a name before it’s been created (by assignment), raises an error

Arithmetic

- a = 5 + 2 # a is 7
- b = 9 – 3. # b is 6.0
- c = 5 * 2 # c is 10
- d = 5**2 # d is 25
- e = 5 % 2 # e is 1

Built in numerical types: int, float, long, complex

- f = 7 / 2
- f = 7 / 2 # f = 3.5
- f = 7 // 2 # f = 3
- f = 7 / 2. # f = 3.5

- f = 7 / float(2) # f is 3.5
- f = int(7 / 2) # f is 3

Complex numbers

- f = 1+2.56j
- print(f.real,f.imag)

String - 1

- Strings can be delimited by matching single or double quotation marks

```
single_quoted_string = 'data science'
double_quoted_string = "data science"
escaped_string = 'Isn\'t this fun'
another_string = "Isn't this fun"

real_long_string = 'this is a really long string. \
It has multiple parts, \
but all in one line.'
```

- Use triple quotes for multi line strings

```
multi_line_string = """This is the first line.
and this is the second line
and this is the third line"""
```

String - 2

- Strings can be concatenated (glued together) with the + operator, and repeated with *

```
s = 3 * 'un' + 'ium' # s is 'unununium'
```

- Two or more *string literals* (i.e. the ones enclosed between quotes) next to each other are automatically concatenated

```
s1 = 'Py' 'thon'
s2 = s1 + '2.7'

real_long_string = ('this is a really long string. '
'It has multiple parts, '
'but all in one line.')
```

Input and Output

```
>>>person = input('Enter your name: ')
>>>print('Hello', person)
```

List - 1

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [ integer_list, heterogeneous_list, [] ]
list_length = len(integer_list) # equals 3
list_sum = sum(integer_list) # equals 6
```

- Get the *i*-th element of a list

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
zero = x[0] # equals 0, lists are 0-indexed
one = x[1] # equals 1
nine = x[-1] # equals 9,  last element
eight = x[-2] # equals 8,  for next-to-last element
```

- Get a slice of a list

```
one_to_four = x[1:5] # [1, 2, 3, 4]
first_three = x[:3] # [0, 1, 2]
last_three = x[-3:] # [7, 8, 9]
three_to_end = x[3:] # [3, 4, ..., 9]
without_first_and_last = x[1:-1] # [1, 2, ..., 8]
copy_of_x = x[:] # [0, 1, 2, ..., 9]
another_copy_of_x = x[:3] + x[3:] # [0, 1, 2, ..., 9]
```

List - 2

- Check for memberships

```
x = 1 in [1, 2, 3] # True
X = 0 in [1, 2, 3] # False
```

- Concatenate lists

```
x = [1, 2, 3]
y = [4, 5, 6]
x.extend(y) # x is now [1,2,3,4,5,6]

x = [1, 2, 3]
y = [4, 5, 6]
z = x + y  # z is [1,2,3,4,5,6]; x is unchanged.
```

- List unpacking (multiple assignment)

```
x, y = [1, 2] # x is 1 and y is 2
[x, y] = 1, 2 # same as above
x, y = [1, 2] # same as above
x, y = 1, 2   # same as above
_, y = [1, 2] # y is 2, didn't care about the first element
```

List - 3

- Modify content of list

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
x[2] = x[2] * 2  # x is [0, 1, 4, 3, 4, 5, 6, 7, 8, 9]
x[-1] = 0 # x is [0, 1, 4, 3, 4, 5, 6, 7, 8, 0]
x[5:8] = []   # x is [0, 1, 4, 3, 4, 8, 0]
del x[:2] # x is [4, 3, 4, 8, 0]
del x[:]  # x is []
del x     # referencing to x hereafter is a NameError
```

- **Strings** can also be sliced. But they cannot modified (they are immutable)

```
s = 'abcdefg'
a = s[0]  # 'a'
x = s[:2] # 'ab'
y = s[-3:]# 'efg'
s[:2] = 'AB'  # this will cause an error
s = 'AB' + s[2:] # str is now ABcdefg
```

Sorting list

- Sorted(list): keeps the original list intact and returns a new sorted list
- list.sort: sort the original list

```
x = [4,1,2,3]
y = sorted(x)      # is [1,2,3,4], x is unchanged
x.sort()           # now x is [1,2,3,4]
```

- Change the default behavior of sorted

```
# sort the list by absolute value from largest to smallest
x = [-4,1,-2,3]
y = sorted(x, key=abs, reverse=True)  # is [-4,3,-2,1]
x.sort(reverse=True)
```

The range() function

range([start], stop[, step])
start: Starting number of the sequence.
stop: Generate numbers up to, but not including this number.
step: Difference between each number in the sequence.

```
for i in range(5):
    print (i) # will print 0, 1, 2, 3, 4 (in separate lines)
for i in range(2, 5):
    print (i) # will print 2, 3, 4
for i in range(0, 10, 2):
    print (i)  # will print 0, 2, 4, 6, 8
for i in range(10, 2, -2):
    print (i)  # will print 10, 8, 6, 4
```

Activity 1

Display the indexes and values of the following list using range,

```
a = ['Mary', 'had', 'a', 'little', 'lamb']

Ex: 0 Mary
    1 had
    …
```

Range() in python 3

- In python 3, range() is an object which can be iterated, but not identical to [0, 1, 2, 3, 4] (lazy iterator)

```
print (range(3)) # in python 3, will see "range(0, 3)"
print (list(range(3))) # will print [0, 1, 2] in python 3

x = range(5)
x[2] = 5  # in python 3, will cause an error.
y = list(x)
y[2] = 5  # [0, 1, 5, 3, 4]
```

Ref to lists

- What are the expected output for the following code?

```
a = list(range(10))
b = a
b[0] = 100
print(a)
```

```
[100, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
a = list(range(10))
b = a[:]
b[0] = 100
print(a)
```

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

tuples

- Similar to lists, but are immutable
- a_tuple = (0, 1, 2, 3, 4)
- Other_tuple = 3, 4
- Another_tuple = tuple([0, 1, 2, 3, 4])
- Hetergeneous_tuple = (‘john’, 1.1, [1, 2])

Note: tuple is defined by comma, not (), which is only used for convenience. So a = (1) is not a tuple,
but a = (1,) is.

- Can be sliced, concatenated, or repeated

```
a_tuple[2:4]    # will print (2, 3)
```

- Cannot be modified

```
a_tuple[2] = 5
```

TypeError: 'tuple' object does not support item assignment

Tuples - 2

- Useful for returning multiple values from functions

```
def sum_and_product(x, y):
    return (x + y),(x * y)
sp = sum_and_product(2, 3)     # equals (5, 6)
s, p = sum_and_product(5, 10) # s is 15, p is 50
```

- Tuples and lists can also be used for multiple assignments

```
x, y = 1, 2
[x, y] = [1, 2]
(x, y) = (1, 2)
x, y = y, x
```

Dictionaries

- A dictionary associates values with unique keys

```
empty_dict = {}                                  # Pythonic
empty_dict2 = dict()                             # less Pythonic
grades = { "Joel" : 80, "Tim" : 95 }             # dictionary literal
```

- Access/modify value with key

```
joels_grade = grades["Joel"]                # equals 80

grades["Tim"] = 99                          # replaces the old value
grades["Kate"] = 100                        # adds a third entry
num_students = len(grades)                  # equals 3
```

Dictionaries - 2

- Check for existence of key

```
joel_has_grade = "Joel" in grades      # True
kate_has_grade = "Kate" in grades      # False
```

- Use “get” to avoid keyError and add default value

```
joels_grade = grades.get("Joel", 0)    # equals 80
kates_grade = grades.get("Kate", 0)    # equals 0
```

- Get all items

```
all_keys = grades.keys()    # return a list of all keys
all_values = grades.values() # return a list of all values
all_pairs = grades.items() # a list of (key, value) tuples
```

Control flow - 1

- if-else

```
if 1 > 2:
    message = "if only 1 were greater than two..."
elif 1 > 3:
    message = "elif stands for 'else if'"
else:
    message = "when all else fails use else (if you want to)"
print (message)
```

Truthiness

All keywords are case sensitive.

0, 0.0, [], (), None are considered False.
Most other values are True.

```
a = [0, 0, 0, 1]

any(a)
Out: True

all(a)
Out: False
```

Comparison

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity
is not	negated object identity

```
a = [0, 1, 2, 3, 4]
b = a
c = a[:]

a == b
Out: True

a is b
Out: True

a == c
Out: True

a is c
Out: False
```

Control flow - 2

- loops

```
x = 0
while x < 10:
    print (x, "is less than 10")
    x += 1
```

What happens if we forgot to indent?

```
for x in range(10):
    pass
```

Keyword pass in loops:
Does nothing, empty statement placeholder

```
for x in range(10):
    if x == 3:
        continue # go immediately to the next iteration
    if x == 5:
        break # quit the loop entirely
    print (x)
```

Functions - 1

- Functions are defined using *def*

```
def double(x):
    """this is where you put an optional docstring
    that explains what the function does.
    for example, this function multiplies its
    input by 2"""
    return x * 2
```

- You can call a function after it is defined

```
z = double(10) # z is 20
```

- You can give default values to parameters

```
def my_print(message="my default message"):
   print (message)

my_print("hello") # prints 'hello'
my_print() # prints 'my default message'
```

Functions - 2

- Sometimes it is useful to specify arguments by name

```
def subtract(a=0, b=0):
    return a - b

subtract(10, 5) # returns 5
subtract(0, 5) # returns -5
subtract(b = 5) # same as above
subtract(b = 5, a = 0) # same as above
```

Functions - 3

- Functions are objects too

```
def double(x):
    print(x * 2)

DD = double
DD(2)
Out[12]: 4
```

```
def apply_to_one(f):
    return f(1)

x=apply_to_one(DD)
Out[16]: 2
```

Lambda functions

lambda argument_list: expression

- lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created.

```
x = lambda a : a + 10
print(x(5))
```

List comprehension

- Lets say, we need to create a list of terms 0-4 squared.

```
squares = []
for x in range(5):
    squares.append(x*x)
print(squares)
Out: [0, 1, 4, 9, 16]
```

- Can right using list comprehension, a very convenient way to create a new list

```
squares = [x * x for x in range(5)]
print(squares)
Out: [0, 1, 4, 9, 16]
```

List comprehension - 2

- Can also be used to filter list

```
even_numbers = [x for x in range(5) if x % 2 == 0]
print(even_numbers)
Out: [0, 2, 4]

even_numbers = []
for x in range(5):
    if x % 2 == 0:
    even_numbers.append(x)
print(even_numbers)
Out: [0, 2, 4]
```

zip

- Useful to combined multiple lists into a list of tuples

```
list(zip(['a', 'b', 'c'], [1, 2, 3], ['A', 'B', 'C']))
Out: [('a', 1, 'A'), ('b', 2, 'B'), ('c', 3, 'C')]
```

```
names = ['James', 'Tom', 'Mary']
grades = [100, 90, 95]
list(zip(names, grades))
Out: [('James', 100), ('Tom', 90), ('Mary', 95)]
```

Argument unpacking

```
gradeBook = [['James', 100],
             ['Tom', 90],
             ['Mary', 95]]
[names, grades]=zip(*gradeBook)

print(names)
Out: ('James', 'Tom', 'Mary')
print(grades)
Out: (100, 90, 95)
```

Module math

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

```
# preferred.
import math
math.abs(-0.5)
```

```
#bad style. Many unknown
#names in name space.
from math import *
abs(-0.5)
```

```
#This is fine
from math import abs
abs(-0.5)
```

Important python modules for data science

- Numpy
  - Key module for scientific computing
  - Convenient and efficient ways to handle multi dimensional arrays
- pandas
  - DataFrame
  - Flexible data structure of labeled tabular data
- Matplotlib: for plotting
- Scipy: solutions to common scientific computing problem such as linear algebra, optimization, statistics, sparse matrix