INTRODUCTION TO PROGRAMMING REVIEW by @htnminh

August 9, 2021

Contents

0 INTRODUCTION TO PROGRAMMING REVIEW

Written by Hoang Tran Nhat Minh,

Hanoi University of Science and Technology,

Data Science and Artificial Intelligence - K65.

Guided by Dr. Dinh Viet Sang.

0.1 PREFACE

Thank you, my teacher, Dr. Dinh Viet Sang, I cannot learn programming in Python that much without you. This notebook uses a huge part your lectures as a guidance.

Dear Data Science & AI - K65,

This is my last notebook in order to prepare for the final exam on Introduction to Programming. If you find this notebook helpful, thanks for reading it. Some might find this one not that helpful, I highly appreciate every line of text you read, including this one.

If you find any mistake, feedback and I will try to fix it as soon as possible.

Thank y'all for supporting me all the time, good luck on all of your exams.

Best wishes,

Hoang Tran Nhat Minh.

0.2 CODES FOR ALL EXERCISES AND SOURCES

SOME OF THE FOLLOWING CONTENTS MIGHT BE ADJUSTED OVER TIME.

Code for all exercises:

https://github.com/htnminh/python-ex-intro-to-prog

Source of this documentation:

https://github.com/htnminh/pdf-python-books-docs/tree/main/PYTHON%20LAST%20REVIEW

Source of this Colab Notebook:

https://colab.research.google.com/drive/1mjtoDbqNHKB2bAb6rTUU8hiJRfUX3Ryt?usp=sharing

1 CHAPTER 1: INTRODUCTION

1.1 CONSTANTS

[1]:
```python
# CONSTANTS
print('Hello World')
print(12.3)
```

```
Hello World
12.3
```

1.2 VARIABLES

[2]:
```python
# VARIABLES
x = 13.4
y = 4.3
x = 'Hello'
print(x)
print(y)
```

```
Hello
4.3
```

1.3 NUMERIC OPERATORS

[3]:
```python
# NUMERIC OPERATORS
print(5 ** 3.5)
print(19 / 5)
print(19 // 5)
print(19 % 5)
```

```
279.5084971874737
3.8
3
4
```

1.4 SCIENTIFIC NOTATION

[4]:
```python
# SCIENTIFIC NOTATION
print(1.234e2)
print(1.234E+2)
print(1.234e-3)
```

```
123.4
123.4
0.001234
```

1.5 INT AND FLOAT

[5]:
```python
# INT AND FLOAT
print(1 + 5.3)
print(5 / 5)
print(3 + 5)
```

```
6.3
1.0
8
```

1.6 TYPES

[6]:
```python
# TYPES
print(3 + 6)
print('hello' + ' there')

print(type(3.5))
print(type('?'))

print(float(3))
print(str(4) + str(5))
print(int('8'))
```

```
9
hello there
<class 'float'>
<class 'str'>
3.0
45
8
```

1.7 (INPUT) TYPE DIFFERENCES

[7]:
```python
# TYPE DIFFERENCES
a = input('a = ')
print(a * 5)
print(int(a) * 5)
```

```
a = 8
88888
40
```

1.8 (INPUT) FILE NAME

[8]:
```python
# FILE NAME
name = input('Name = ')
#/content/sample_data/README.md
f = open(name, 'r')
print(f.read())
f.close()
```

```
Name = /content/sample_data/README.md
This directory includes a few sample datasets to get you started.

*   `california_housing_data*.csv` is California housing data from the 1990 US
    Census; more information is available at:
    https://developers.google.com/machine-learning/crash-course/california-
housing-data-description
```

```
*   `mnist_*.csv` is a small sample of the
    [MNIST database](https://en.wikipedia.org/wiki/MNIST_database), which is
    described at: http://yann.lecun.com/exdb/mnist/

*   `anscombe.json` contains a copy of
    [Anscombe's quartet](https://en.wikipedia.org/wiki/Anscombe%27s_quartet); it
    was originally described in

    Anscombe, F. J. (1973). 'Graphs in Statistical Analysis'. American
    Statistician. 27 (1): 17-21. JSTOR 2682899.

    and our copy was prepared by the
    [vega_datasets library](https://github.com/altair-viz/vega_datasets/blob/4f6
7bdaad10f45e3549984e17e1b3088c731503d/vega_datasets/_data/anscombe.json).
```

2 CHAPTER 2: CONTROL FLOW

2.1 COMPARISON OPERATORS

[9]:
```python
# COMPARISON OPERATORS
print(3 == 4)
print(3 != 4)
print(3 < 4)
print(3 >= 4)
```

```
False
True
True
False
```

2.2 LOGIC OPERATORS

[10]:
```python
# LOGIC OPERATORS
print(not False)
print(True and False)
print(True or False)
```

```
True
False
True
```

2.3 (INPUT) BRANCHING

[11]:
```python
# BRANCHING
a = int(input('a = '))
b = int(input('b = '))
if a > b:
    print('larger')
elif a == b:
    print('equal')
else:
    print('less')
```

```
a = 3
b = 5
less
```

3 CHAPTER 3: FUNCTIONS

3.1 FUNCTION EXAMPLE

[12]:
```
# FUNCTION EXAMPLE

#   name    parameter(s)
def two_time(number):
    # docstring
    '''
    2 times a number
    '''
    # body
    doubled = 2 * number
    # returns (if not, it is a void function)
    return doubled
# function call, pass argument(s) to parameter(s)
print(two_time(8))

print(two_time.__doc__)
```

```
16

    2 times a number

```

3.2 COMMON BUILT-IN PYTHON FUNCTIONS

[13]:
```
# COMMON BUILT-IN PYTHON FUNCTIONS
# previous ones: input(), type(), float(),...
print(max(6, 8, 5))
print(min(3, 6, 7))
```

```
8
3
```

3.3 PASS-BY-OBJECT-REFERENCE

[14]:
```
# PASS-BY-OBJECT-REFERENCE
# Immutable objects: int, float, complex, string, tuple, frozen set, bytes
# Mutable objects: list, dict, set, byte array

def change_num(num):
    num += 1
a = 4
print(a)
change_num(a)
print(a)

def change_lst(lst):
    lst.append('changed')
l = ['original']
print(l)
change_lst(l)
print(l)
```

```
4
4
['original']
['original', 'changed']
```

3.4 SCOPE

[15]:
```
# SCOPE
def f(x):
    x += 1
    print('In f:    x =',x)
x = 0
print('First:   x =', x)
f(x)
print('After f: x =', x)
```

```
First:   x = 0
In f:    x = 1
After f: x = 0
```

3.5 FUNCTIONS AS ARGUMENTS

[16]:
```
# FUNCTIONS AS ARGUMENTS
def triple(num):
    return 3 * num
def square(num):
    return num ** 2

def fx_plus_gy(f, x, g, y):
    return f(x) + g(y)

print(fx_plus_gy(triple, 5, square, 8))
print(3 * 5 + 8 ** 2)
```

```
79
79
```

3.6 DEFAULT PARAMETER VALUE

[17]:
```
# DEFAULT PARAMETER VALUE
def tong(a, b = 2, c = 3):
    return a + b + c

print(tong(1))
# 1 + 2 + 3
print(tong(1, 4))
# 1 + 4 + 3
print(tong(0, 3, 6))
# 0 + 3 + 6
```

```
6
8
9
```

3.7 LAMBDA FUNCTION

[18]:
```
# LAMBDA FUNCTION
tong = lambda x, y: x + y
print(tong(3, 5))
print((lambda x, y: x * y)(2, 9))
```

```
8
18
```

3.7.1 LAMBDA EXAMPLE

[19]:
```
# LAMBDA EXAMPLE
def multiply_by(n):
    return lambda x: x * n

double = multiply_by(2)
# multiply_by(2) ->  lambda x: x * 2
# -> a doubling function
triple = multiply_by(3)

print(double(16))
print(triple(4))
```

```
32
12
```

3.7.2 TEST: WITHOUT LAMBDA

[20]:
```python
# TEST: WITHOUT LAMBDA
def multiply_by(n):
    def multi(x):
        return x * n
    return multi

double = multiply_by(2)
# multiply_by(2) -> function: multi, where multi return the doubled argument
# -> a doubling function
triple = multiply_by(3)

print(double(16))
print(triple(4))
```

```
32
12
```

3.7.3 LAMBDA EXAMPLE

[21]:
```python
# LAMBDA EXAMPLE
x_plus_fx = lambda x, f: x + f(x)

print(x_plus_fx(3, lambda x: x ** 2))
# x + f(x) = x + x ** 2
# 3 + f(3) = 3 + 3 ** 2
```

```
12
```

3.8 RECURSION EXAMPLES

[22]:
```python
# RECURSION EXAMPLES
def pr(n):
    print(str(n) + '! = ' + str(n - 1) + '! * ' + str(n))

def factorial(n):
    if n == 0:
        print('0! = 1')
        return 1
    pr(n)
    return factorial(n - 1) * n

print(factorial(7))
```

```
7! = 6! * 7
6! = 5! * 6
5! = 4! * 5
4! = 3! * 4
3! = 2! * 3
2! = 1! * 2
1! = 0! * 1
0! = 1
5040
```

3.8.1 FACTORIAL RECURSION IN SHORT

[23]:
```
# FACTORIAL RECURSION IN SHORT
factorial = lambda n: 1 if n == 0 else factorial(n - 1) * n
print(factorial(7))
```

```
5040
```

3.8.2 FIBONACCI RECURSION IN SHORT

[24]:
```
# FIBONACCI RECURSION IN SHORT
fib = lambda n: n if n <= 1 else fib(n - 1) + fib(n - 2)
print(*[fib(i) for i in range(15)])
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

3.8.3 (INPUT) (WARNING: WALL-OF-CODE) TOWER OF HANOI: TRADITIONAL RECURSION PROBLEM

[25]:
```
# TOWER OF HANOI: TRADITIONAL RECURSION PROBLEM
# EXPLICITLY VISUALIZED CODE
def hanoi_tower(n):
    '''run and return number of transfers as text'''
    def transfer(n, start, end, mid):
        if n == 1:
            nonlocal count
            count += 1
            map_ind = ['A', 'B', 'C']
            print(' '*18 + 'Step ' + str(count) + ': ' + map_ind[start] + ' ->␣
↪' + map_ind[end])
            move(start, end)
            print_board()
        else:
            transfer(n - 1, start, mid, end)
            transfer(1, start, end, mid)
            transfer(n - 1, mid, end, start)
    count = 0
    transfer(n, 0, 2, 1)
    return count
```

```
'''
a =
      0  1  2  3

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

printed :
                A  B  C

                1  .  .
                2  .  .
                3  .  .
                4  .  .
'''
def print_board():
    '''print current board'''
    print('|  A  B  C  |')

    for col in range(n - 1, -1, -1):
        print('|  ', end = '')
        for row in range(3):
            character = '.' if a[row][col] == 0 else a[row][col]
            print(str(character) + '  ', end = '')
        print('|')


def move(row_start, row_end):
    '''make a move'''
    def col_lastnonenull(row):
        for col_ind in range(n - 1, -1, -1):
            if a[row][col_ind] != 0:
                return col_ind
            # return n - 1
    def col_firstnull(row):
        for col_ind in range(n):
            if a[row][col_ind] == 0:
                return col_ind
            # return 0
    a[row_end][col_firstnull(row_end)] =␣
 ↪a[row_start][col_lastnonenull(row_start)]
    a[row_start][col_lastnonenull(row_start)] = 0

# MAIN:
```

```python
# input n
n = int(input('n = '))

# initialize list of list, n = 4,
'''
a =
      0  1  2  3

0     4  3  2  1
1     0  0  0  0
2     0  0  0  0
'''
a = [[n - i for i in range(n)]]
for i in range(2):
    a.append([0 for j in range(n)])

# print the initialized board
print()
print_board()

# run, print the board every move and return the result
print('\nNumber of transfers: %i' % hanoi_tower(n))

# n SHOULD BE LESS THAN 5
```

```
n = 4

|  A  B  C  |
|  1  .  .  |
|  2  .  .  |
|  3  .  .  |
|  4  .  .  |
                  Step 1: A -> B
|  A  B  C  |
|  .  .  .  |
|  2  .  .  |
|  3  .  .  |
|  4  1  .  |
                  Step 2: A -> C
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|  3  .  .  |
|  4  1  2  |
                  Step 3: B -> C
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|  3  .  1  |
|  4  .  2  |
                  Step 4: A -> B
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|  .  .  1  |
|  4  3  2  |
                  Step 5: C -> A
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|  1  .  .  |
|  4  3  2  |
                  Step 6: C -> B
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|  1  2  .  |
|  4  3  .  |
                  Step 7: A -> B
|  A  B  C  |
|  .  .  .  |
|  .  1  .  |
|  .  2  .  |
|  4  3  .  |
                  Step 8: A -> C
|  A  B  C  |
|  .  .  .  |
|  .  1  .  |
|  .  2  .  |
|  .  3  4  |
                  Step 9: B -> C
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|  .  2  1  |
|  .  3  4  |
                  Step 10: B -> A
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|  .  .  1  |
|  2  3  4  |
                  Step 11: C -> A
|  A  B  C  |
|  .  .  .  |
|  .  .  .  |
|   1   .   .   |
|   2   3   4   |
                    Step 12: B -> C
|   A   B   C   |
|   .   .   .   |
|   .   .   .   |
|   1   .   3   |
|   2   .   4   |
                    Step 13: A -> B
|   A   B   C   |
|   .   .   .   |
|   .   .   .   |
|   .   .   3   |
|   2   1   4   |
                    Step 14: A -> C
|   A   B   C   |
|   .   .   .   |
|   .   .   2   |
|   .   .   3   |
|   .   1   4   |
                    Step 15: B -> C
|   A   B   C   |
|   .   .   1   |
|   .   .   2   |
|   .   .   3   |
|   .   .   4   |

Number of transfers: 15
```

4 CHAPTER 4: STRINGS

4.1 STRING EXAMPLES

[26]:
```python
# STRING EXAMPLES
print('abc')
```

```
abc
```

4.2 len() OF A STRING

[27]:
```python
# LEN OF A STRING
print(len('this is a string'))
```

```
16
```

4.3 LOOP OVER A STRING

[28]:
```python
# LOOP OVER A STRING
for c in 'hi there?':
    print(c)
```

```
h
i

t
h
e
r
e
?
```

4.4 SLICING A STRING

[29]:
```python
# SLICING A STRING
print('abcd'[2])
print('mnpq'[1:])
print('hello from the other side'[:5])
```

```
c
npq
hello
```

4.5 STRING FORMATTING OPERATOR %

[30]:
```python
# STRING FORMATTING OPERATOR %
print('%s is a metal' % 'gold')

color = 'yellow'
num = 7
print( '%s is a color of %d main colors in a rainbow!' % (color, num) )

# %s string | %c char | %d decimal | %i integer | %f float
```

```
gold is a metal
yellow is a color of 7 main colors in a rainbow!
```

4.5.1 STRING FORMATTING OPERATOR % EXAMPLE

[31]:
```python
# STRING FORMATTING OPERATOR % EXAMPLE
# https://docs.python.org/3/library/string.html
print('%.2f' % 3.895)
```

```
3.90
```

4.6 in OPERATOR

[32]:
```
# in OPERATOR
print('e' in 'hello')
```

```
True
```

4.7 STRING COMPARISON

[33]:
```
# STRING COMPARISON
print('abc' > 'def')
print('abc' > 'd')
print('axyz' < 'bcde')
```

```
False
False
True
```

4.8 THE DIRECTORY FUNCTION dir()

[34]:
```
# THE DIRECTORY FUNCTION dir()
s = '????'
print('...',*dir(s)[30:38], '...\nand a lot more!')
```

```
… __sizeof__ __str__ __subclasshook__ capitalize casefold center count encode
…
and a lot more!
```

4.9 IMPORTANT FUNCTIONS AND METHODS

4.9.1 capitalize(), center()

[35]:
```
# capitalize(), center()
print('hello'.capitalize())
print('hello'.center(14))
print('hello'.center(20, '*'))
```

```
Hello
    hello
*******hello********
```

4.9.2 endswith(), startswith()

[36]:
```
# endswith(), startswith()
print('hello'.endswith('o'))
print('hello'.endswith('o', 1, 4))
#       01234
# search from 1 to 3 = 'ell'
print()
print('hello'.startswith('h'))
print('hello'.startswith('e', 1, 4))
#      01234
# search from 1 to 3 = 'ell'
```

```
True
False

True
True
```

4.9.3 find()

[37]:
```
# find()
print('hello'.find('l'))
print('hello'.find('l', 3, 5))
#      01234
```

```
2
3
```

4.9.4 lstrip(), rstrip(), strip()

[38]:
```
# lstrip(), rstrip(), strip()
def pr(s):
    print('"%s"' % s)

pr('    hello        '.lstrip())
pr('***__   hello __*****  '.lstrip('*'))
print()

pr('    hello        '.rstrip())
pr('***__   hello __*****  '.rstrip('*'))
print()

pr('    hello        '.strip())
pr('***__   hello __*****  '.strip('*'))
print()
```

```
"hello        "
"__   hello __*****  "

"    hello"
"***__   hello __*****  "

"hello"
"__   hello __*****  "
```

4.9.5 join()

[39]:
```python
# join()
wrdlst = ['hello', 'from', 'the', 'other', 'side']
print('_'.join(wrdlst))

wrdset = {'hello', 'from', 'the', 'other', 'side'}
print('-'.join(wrdset))
```

```
hello_from_the_other_side
side-other-the-from-hello
```

4.9.6 replace()

[40]:
```python
# replace()
print('hello, long time no see'.replace('l', '*'))
print('hello, long time no see'.replace('l', '*', 2))
# only the 2 first ones are replaced
```

```
he**o, *ong time no see
he**o, long time no see
```

4.9.7 lower(), upper()

[41]:
```python
# lower(), upper()
print('hELlO'.lower())
print('hELlO'.upper())
```

```
hello
HELLO
```

5 CHAPTER 5: LISTS, SETS, DICTIONARIES AND TUPLES

5.1 LISTS

5.1.1 LIST EXAMPLE

[42]:
```python
# LIST EXAMPLE
lst = ['a', 8, 'bc', 'd', [15, 6]]
print(lst)
```

```
['a', 8, 'bc', 'd', [15, 6]]
```

5.1.2 LOOP A LIST

[43]:
```python
# LIST LOOPS
for ele in lst:
    print(ele)
```

```
a
8
bc
d
[15, 6]
```

5.1.3 LISTS ARE MUTABLE

[44]:
```python
# LISTS ARE MUTABLE
lst[0] = '^_^'
print(lst)
```

```
['^_^', 8, 'bc', 'd', [15, 6]]
```

5.1.4 len() OF A LIST

[45]:
```python
# len OF A LIST
print(len(lst))
print(lst.__len__())
```

```
5
5
```

5.1.5 THE range() FUNCTION

[46]:
```python
# THE range() FUNCTION
print(*range(8))
print(*range(3, 8))
print(*range(3, 8, 3))
print(*range(8, 3, -2))
```

```
0 1 2 3 4 5 6 7
3 4 5 6 7
3 6
8 6 4
```

5.1.6 CONCATENATING LISTS USING +

[47]:
```python
# CONCATENATING LISTS USING +
print(lst)
print(lst + [7, 3, 'h'])
```

```
['^_^', 8, 'bc', 'd', [15, 6]]
['^_^', 8, 'bc', 'd', [15, 6], 7, 3, 'h']
```

5.1.7 LIST SLICING

[48]:
```
# LIST SLICING
print(lst)
print(lst[:3])
print(lst[-2])
```

```
['^_^', 8, 'bc', 'd', [15, 6]]
['^_^', 8, 'bc']
d
```

5.1.8 LIST METHODS AND FUNCTIONS

[49]:
```
# LIST METHODS AND FUNCTIONS
for met in dir(lst)[34:]:
    print(met)
```

```
__subclasshook__
append
clear
copy
count
extend
index
insert
pop
remove
reverse
sort
```

append()

[50]:
```
# append()
new_lst = lst[:]
new_lst.append('appended string')
print(new_lst)
```

```
['^_^', 8, 'bc', 'd', [15, 6], 'appended string']
```

in OPERATOR

[51]:
```
# in OPERATOR
print(8 in lst)
print(9 in lst)
print('^_^' in lst)
print('^_^' not in lst)
```

```
True
False
True
False
```

max(), min(), sum()

[52]:
```
# max(), min(), sum()
new_lst = [6, 69, 9]
print(max(new_lst))
print(min(new_lst))
print(sum(new_lst))
```

```
69
6
84
```

sort(), sorted()

[53]:
```
# sort(), sorted()
new_lst = [6, 69, 9]
new_lst = sorted(new_lst)
print(new_lst)
new_lst.sort(reverse = True, key = lambda x: x % 8)
'''
6 % 8 = 6
69 % 8 = 5
9 % 8 = 1
'''
print(new_lst)
```

```
[6, 9, 69]
[6, 69, 9]
```

del, pop(), remove()

[54]:
```
# del, pop(), remove()
new_lst = lst[:]
print(new_lst)

del new_lst[4]
print(new_lst)

new_lst.pop()
print(new_lst)

new_lst.remove('bc')
print(new_lst)
```

```
['^_^', 8, 'bc', 'd', [15, 6]]
['^_^', 8, 'bc', 'd']
['^_^', 8, 'bc']
['^_^', 8]
```

reverse()

[55]:
```
# reverse()
new_lst = lst[:]
print(new_lst)

new_lst.reverse()
print(new_lst)
```

```
['^_^', 8, 'bc', 'd', [15, 6]]
[[15, 6], 'd', 'bc', 8, '^_^']
```

5.1.9 LISTS AND STRINGS

[56]:
```
# LISTS AND STRINGS
s = 'this one! is a string'
print(list(s))
print(s.split())
print(s.split('!'))

new_lst = ['hi', 'there', '?']
print('-'.join(new_lst))
```

```
['t', 'h', 'i', 's', ' ', 'o', 'n', 'e', '!', ' ', 'i', 's', ' ', 'a', ' ', 's',
't', 'r', 'i', 'n', 'g']
['this', 'one!', 'is', 'a', 'string']
['this one', ' is a string']
hi-there-?
```

5.1.10 ALIASES

[57]:
```
# ALIASES
lst_a = [1, 3, 4]
lst_b = lst_a

lst_b.append('?')
print(lst_a)
```

```
[1, 3, 4, '?']
```

5.1.11 MUTATION AND ITERATION

[58]:
```
# MUTATION AND ITERATION
lst_a = [1, 3, 4]
lst_b = [4, 3, 9]
# trying to remove values in lst_a which are already in lst_b
for num in lst_a:
    if num in lst_b:
        lst_a.remove(num)
print(lst_a)
# avoid mutating the list while iterating over it
```

```
[1, 4]
```

5.1.12 LIST ARGUMENTS

[59]:
```
# LIST ARGUMENTS
def delete_last(a_lst):
    del a_lst[-1]
    # this function mutates the global list

def wrong_delete_last(a_lst):
    a_lst = a_lst[:-1]
    print('List in wrong one:', a_lst)
    # this one doesn't, a_lst now becomes a local variable

new_lst = [1, 3, 6, 9]
delete_last(new_lst)
print(new_lst)

new_lst = [1, 3, 6, 9]
wrong_delete_last(new_lst)
print(new_lst)
```

```
[1, 3, 6]
List in wrong one: [1, 3, 6]
[1, 3, 6, 9]
```

5.1.13 MapReduce AND LIST COMPREHENSION

map()

[60]:
```
# map()
new_lst = [1, 3, 6, 8, 9, 10]
print(list(map(lambda x: x + 3, new_lst)))
```

```
[4, 6, 9, 11, 12, 13]
```

reduce()

[61]:
```
# reduce()
from functools import reduce
new_lst = [1, 3, 6, 1]
print(reduce(lambda x, y: x + y, new_lst))
print(reduce(lambda x, y: x + y, new_lst, 3000))
#                                         initializer
```

```
11
3011
```

MapReduce APPLICATION: COUNT NUMBER OF A WORD

[62]:
```
# MapReduce APPLICATION: COUNT NUMBER OF A WORD
# count the number of the word "the"/"The" in a sentence
```

```
text = 'The word "deep" in "deep learning" refers to the number of layers␣
 ↪through which the data is transformed. More precisely, deep learning systems␣
 ↪have a substantial credit assignment path (CAP) depth. The CAP is the chain␣
 ↪of transformations from input to output. CAPs describe potentially causal␣
 ↪connections between input and output. For a feedforward neural network, the␣
 ↪depth of the CAPs is that of the network and is the number of hidden layers␣
 ↪plus one (as the output layer is also parameterized). For recurrent neural␣
 ↪networks, in which a signal may propagate through a layer more than once,␣
 ↪the CAP depth is potentially unlimited.[2] No universally agreed-upon␣
 ↪threshold of depth divides shallow learning from deep learning, but most␣
 ↪researchers agree that deep learning involves CAP depth higher than 2. CAP␣
 ↪of depth 2 has been shown to be a universal approximator in the sense that␣
 ↪it can emulate any function.[15] Beyond that, more layers do not add to the␣
 ↪function approximator ability of the network. Deep models (CAP > 2) are able␣
 ↪to extract better features than shallow models and hence, extra layers help␣
 ↪in learning the features effectively.'
ifThe = lambda word: 1 if word in ['the', 'The'] else 0
sumof2 = lambda x, y: x + y
print(list(map(ifThe, text.split())))
print(reduce(sumof2, list(map(ifThe, text.split()))))
```

```
[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
15
```

LIST COMPREHENSION

[63]:
```
# LIST COMPREHENSION
print([c for c in 'hello there'], '\n')

cnt_lst = [ word for word in text.split() if word in ['the', 'The'] ]
print(cnt_lst)
print(len(cnt_lst))

cnt_lst_2 = [ 1 if word in ['the', 'The'] else 0 for word in text.split() ]
print(cnt_lst_2)
```

```
['h', 'e', 'l', 'l', 'o', ' ', 't', 'h', 'e', 'r', 'e']

['The', 'the', 'the', 'The', 'the', 'the', 'the', 'the', 'the', 'the', 'the',
'the', 'the', 'the', 'the']
15
[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
```

5.2 SETS

5.2.1 SET EXAMPLES

[64]:
```
# SET EXAMPLES
print({3, 4, 8})
print(set('hello'))
```

```
{8, 3, 4}
{'l', 'o', 'h', 'e'}
```

5.2.2 SET OPERATIONS

[65]:
```
# SET OPERATIONS
A = {0, 1, 2, 3}
B = {2, 3, 4, 5}
print(A - B) # in A, not in B
print(A | B) # in any of A or B (bitwise or)
print(A & B) # in both A and B (bitwise and)
print(A ^ B) # in A or B, not both (bitwise xor)
```

```
{0, 1}
{0, 1, 2, 3, 4, 5}
{2, 3}
{0, 1, 4, 5}
```

5.2.3 SET COMPREHENSION

[66]:
```
# SET COMPREHENSION
# set of all end-of-sentence words
print( {word for word in text.split() if word.endswith('.')} )
```

```
{'effectively.', 'output.', 'transformed.', 'network.', 'depth.', '2.',
'parameterized).'}
```

5.3 DICTIONARIES

5.3.1 DICTIONARY EXAMPLES

[67]:
```
# DICTIONARY EXAMPLES
d = {8: 'b', '?': 'a'}
print(d)
print(d[8])
print(d['?'])
d['!?'] = 'mnpq'
print(d)
```

```
{8: 'b', '?': 'a'}
b
a
{8: 'b', '?': 'a', '!?': 'mnpq'}
```

5.3.2 in OPERATOR

[68]:
```
# in OPERATOR
d = {8: 'b', '?': 'a'}
print(d)
print('?' in d)
print('b' in d)
```

```
{8: 'b', '?': 'a'}
True
False
```

5.3.3 get() METHOD

[69]:
```
# get() METHOD
d = {8: 'b', '?': 'a'}
print(d)
print(d.get(8))
print(d.get('a'))
print(d.get('m', 5)) # not equivalent to d['m'] = 5
print(d)
```

```
{8: 'b', '?': 'a'}
b
None
5
{8: 'b', '?': 'a'}
```

5.3.4 APPLICATION: COUNTING WORDS

[70]:
```
# APPLICATION: COUNTING WORDS
counted = dict()
for word in text.split():
    counted[word] = counted.get(word, 0) + 1
print(counted)
```

```
{'The': 2, 'word': 1, '"deep"': 1, 'in': 4, '"deep': 1, 'learning"': 1,
'refers': 1, 'to': 5, 'the': 13, 'number': 2, 'of': 8, 'layers': 4, 'through':
2, 'which': 2, 'data': 1, 'is': 6, 'transformed.': 1, 'More': 1, 'precisely,':
```

```
1, 'deep': 3, 'learning': 4, 'systems': 1, 'have': 1, 'a': 5, 'substantial': 1,
'credit': 1, 'assignment': 1, 'path': 1, '(CAP)': 1, 'depth.': 1, 'CAP': 4,
'chain': 1, 'transformations': 1, 'from': 2, 'input': 2, 'output.': 2, 'CAPs':
2, 'describe': 1, 'potentially': 2, 'causal': 1, 'connections': 1, 'between': 1,
'and': 3, 'For': 2, 'feedforward': 1, 'neural': 2, 'network,': 1, 'depth': 5,
'that': 3, 'network': 1, 'hidden': 1, 'plus': 1, 'one': 1, '(as': 1, 'output':
1, 'layer': 2, 'also': 1, 'parameterized).': 1, 'recurrent': 1, 'networks,': 1,
'signal': 1, 'may': 1, 'propagate': 1, 'more': 2, 'than': 3, 'once,': 1,
'unlimited.[2]': 1, 'No': 1, 'universally': 1, 'agreed-upon': 1, 'threshold': 1,
'divides': 1, 'shallow': 2, 'learning,': 1, 'but': 1, 'most': 1, 'researchers':
1, 'agree': 1, 'involves': 1, 'higher': 1, '2.': 1, '2': 1, 'has': 1, 'been': 1,
'shown': 1, 'be': 1, 'universal': 1, 'approximator': 2, 'sense': 1, 'it': 1,
'can': 1, 'emulate': 1, 'any': 1, 'function.[15]': 1, 'Beyond': 1, 'that,': 1,
'do': 1, 'not': 1, 'add': 1, 'function': 1, 'ability': 1, 'network.': 1, 'Deep':
1, 'models': 2, '(CAP': 1, '>': 1, '2)': 1, 'are': 1, 'able': 1, 'extract': 1,
'better': 1, 'features': 2, 'hence,': 1, 'extra': 1, 'help': 1, 'effectively.':
1}
```

5.3.5 LOOP OVER A DICTIONARY

[71]:
```python
# LOOP OVER A DICTIONARY
d = {8: 'b', '?': 'a', '!': 'b', 10: 15}
for key in d:
    print(key, d[key])
```

```
8 b
? a
! b
10 15
```

5.3.6 LISTS OF KEYS AND VALUES

[72]:
```python
# LISTS OF KEYS AND VALUES
print(list(d.keys()))
print(list(d.values()))
print(list(d.items()))
```

```
[8, '?', '!', 10]
['b', 'a', 'b', 15]
[(8, 'b'), ('?', 'a'), ('!', 'b'), (10, 15)]
```

5.3.7 TWO ITERATION VARIABLES

[73]:
```python
# TWO ITERATION VARIABLES
for key, value in d.items():
    print(key, value)
```

```
8 b
? a
```

```
! b
10 15
```

5.3.8 DICTIONARY COMPREHENSION

[74]:
```
# DICTIONARY COMPREHENSION
print({x: x * 8 for x in range(2, 8)})
```

```
{2: 16, 3: 24, 4: 32, 5: 40, 6: 48, 7: 56}
```

5.3.9 MEMOIZED RECURSION: FIBONACCI EXAMPLE

[75]:
```
# MEMOIZED RECURSION: FIBONACCI EXAMPLE
res = {0: 0, 1: 1}
def fib(n):
    if n in res:
        return res[n]
    res[n] = fib(n - 1) + fib(n - 2)
    return res[n]
print(fib(14))
```

```
377
```

MEMOIZED RECURSION: LONGEST INCREASING SUBSEQUENCE EXAMPLE

[76]:
```
# MEMOIZED RECURSION: LONGEST INCREASING SUBSEQUENCE EXAMPLE
from random import randint
num_list = [randint(0, 100) for i in range(15)]

res = {0: 1}
def lis_including(i):
    if i in res:
        return res[i]
    max_lis = 1
    for j in range(i):
        if num_list[j] < num_list[i]:
            max_lis = max(max_lis, 1 + lis_including(j))
    res[i] = max_lis
    return max_lis

print('  i   num_list[i]    lis_including(i)')
for i in range(15):
    print('%3i%13i%20i' % (i, num_list[i], lis_including(i)))
print('Result =', max([lis_including(i) for i in range(15)]))
```

```
  i   num_list[i]    lis_including(i)
  0           96                   1
  1            9                   1
  2           69                   2
  3           50                   2
  4           23                    2
  5           89                    3
  6           31                    3
  7           64                    4
  8           11                    2
  9           99                    5
 10           79                    5
 11           98                    6
 12           18                    3
 13           35                    4
 14           55                    5
Result = 6
```

5.4 TUPLES

5.4.1 TUPLE EXAMPLES

[77]:
```python
# TUPLE EXAMPLES
print((3, 5, 7))
print((1, ))
# Warning: Tuple of 1 element must have a comma
print((1))
```

```
(3, 5, 7)
(1,)
1
```

5.4.2 TUPLES ARE IMMUTABLE

[78]:
```python
# TUPLES ARE IMMUTABLE
tup = (3, 4, 7)
# tup[0] = 0
# TypeError: 'tuple' object does not support item assignment
print(tup)
```

```
(3, 4, 7)
```

5.4.3 TUPLE DIRECTORIES

[79]:
```python
# TUPLE DIRECTORIES
print(dir(tuple)[-3:])
```

```
['__subclasshook__', 'count', 'index']
```

5.4.4 TUPLES AND ASSIGNMENT

[80]:
```python
# TUPLES AND ASSIGNMENT
a, b = (3, 5)
print('a = %i\nb = %i' % (a, b))
x, y = 6, 9
```

```
print('x = %i\ny = %i' % (x, y))
```

```
a = 3
b = 5
x = 6
y = 9
```

5.4.5 TUPLE AS RETURN VALUES

[81]:
```
# TUPLE AS RETURN VALUES
print(divmod(100, 32))

def max_and_index(lst):
    index = 0
    for i in range(1, 15):
        if lst[i] > lst[index]:
            index = i
    return lst[index], index


from random import randint
num_list = [randint(0, 100) for i in range(15)]
print(num_list)
print(max_and_index(num_list))
```

```
(3, 4)
[46, 23, 33, 7, 14, 96, 42, 48, 37, 31, 85, 1, 21, 28, 32]
(96, 5)
```

5.4.6 LISTS AND TUPLES

[82]:
```
# LISTS AND TUPLES
z = zip('abcde', [3, 4, 5, 6, 7, 8, 9, 10])
print(z)
z = zip('abcde', [3, 4, 5, 6, 7, 8, 9, 10])
print(list(z))
z = zip('abcde', [3, 4, 5, 6, 7, 8, 9, 10])
for pair in z:
    print(pair)
```

```
<zip object at 0x7f39bfea34b0>
[('a', 3), ('b', 4), ('c', 5), ('d', 6), ('e', 7)]
('a', 3)
('b', 4)
('c', 5)
('d', 6)
('e', 7)
```

`enumerate()`

[83]:
```
# enumerate()
for ind, c in enumerate('cdef'):
    print(ind, c)
```

```
0 c
1 d
2 e
3 f
```

5.4.7 DICTIONARIES AND TUPLES

[84]:
```
# DICTIONARIES AND TUPLES
d = {'h': 4, 't': 5, 'n': 6, 'm': 7}
print(d.items())
l = [(8, 'h'), (7, 't'), (6, 'n'), (5, 'm')]
print(dict(l))
```

```
dict_items([('h', 4), ('t', 5), ('n', 6), ('m', 7)])
{8: 'h', 7: 't', 6: 'n', 5: 'm'}
```

5.4.8 TUPLE COMPARISON

[85]:
```
# TUPLE COMPARISON
print((3, ) < (5, ))
print((3, 6) < (5, 1))
print((3, 6) < (3, 2))
print()
print(('Alpha', 0) < ('Beta', 8))
print(('Alpha', 0) < ('Alpha', -2))
```

```
True
True
False

True
False
```

5.4.9 SORTING EXAMPLE: SORT BY VALUE, NOT KEY

[86]:
```
# SORTING EXAMPLE: SORT BY VALUE, NOT KEY
d = {'h': 4, 't': 5, 'n': 6, 'm': 7}
print('Sort by key:          ', *sorted(list(d.items())))

print('Sort by value, #1 way:', *sorted(list(d.items()), key = lambda x: x[1]))

value_key_list = sorted([(value, key) for key, value in d.items()])
sorted_value_list = [(key, value) for value, key in value_key_list]
print('Sort by value, #2 way:', *sorted_value_list)
```

```
Sort by key:           ('h', 4) ('m', 7) ('n', 6) ('t', 5)
Sort by value, #1 way: ('h', 4) ('t', 5) ('n', 6) ('m', 7)
Sort by value, #2 way: ('h', 4) ('t', 5) ('n', 6) ('m', 7)
```

6 CHAPTER 6: MODULES

6.1 import EXAMPLE

[87]:
```
# import EXAMPLE
import math
print(math.sqrt(8))
from math import sqrt
print(sqrt(8))
from random import *
print(randint(3,6))
```

```
2.8284271247461903
2.8284271247461903
5
```

6.2 (preparation)

6.2.1 (preparation) mount

[88]:
```
# (preparation) mount
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

6.2.2 (preparation) append path

[89]:
```
# (preparation) append path
!ls /content/gdrive/MyDrive/Colab\ Notebooks/hello.py
!cat /content/gdrive/MyDrive/Colab\ Notebooks/hello.py
import sys
sys.path.append('/content/gdrive/MyDrive/Colab Notebooks')
```

```
'/content/gdrive/MyDrive/Colab Notebooks/hello.py'
if __name__ == '__main__':
    print('Hello?')
```

6.3 MODULES AS SCRIPTS

[90]:
```
# MODULES AS SCRIPTS
hello_path = '/content/gdrive/MyDrive/Colab Notebooks/hello.py'

print('Script in "hello.py":\n---------------------------------')
f = open(hello_path, 'r')
```

```
print(f.read(), '\n---------------------------------')
f.close()

! python '/content/gdrive/MyDrive/Colab Notebooks/hello.py'
```

```
Script in "hello.py":
---------------------------------
if __name__ == '__main__':
    print('Hello?')
---------------------------------
Hello?
```

6.4 __name__

[91]:
```
# __name__
print('before import')
import hello
print('after import')
print(hello.__name__)
```

```
before import
after import
hello
```

6.5 sys.argv[]

[92]:
```
# sys.argv[]
sum_path = '/content/gdrive/MyDrive/Colab Notebooks/sumof2module.py'
print('Script in "sumof2module.py":\n---------------------------------')
f = open(sum_path, 'r')
print(f.read(), '\n---------------------------------')
f.close()
! python '/content/gdrive/MyDrive/Colab Notebooks/sumof2module.py' 8 9
```

```
Script in "sumof2module.py":
---------------------------------
if __name__ == '__main__':
    import sys
    print(int(sys.argv[1]) + int(sys.argv[2]))
---------------------------------
17
```

6.6 USEFUL MODULES: BASIC EXAMPLES

[93]:
```
# USEFUL MODULES: BASIC EXAMPLES
# See a lot more in-depth examples in the APPENDIX
''' Find out more in: (ctrl + click to open)
https://docs.python.org/3/library/random.html
```

```
https://docs.python.org/3/library/datetime.html
https://docs.python.org/3/library/math.html
https://numpy.org/doc/
'''

import random
import datetime
import math
import numpy

print(datetime.datetime.now())
print(math.factorial(8))

np_arr = numpy.array([random.uniform(0, 3) for i in range(6)])
print(np_arr)
# print values which are > 1.5
print(np_arr[np_arr > 1.5])
```

```
2021-08-09 05:48:01.231481
40320
[1.79077237 1.05060928 1.15447232 2.275137   1.86980938 1.18032189]
[1.79077237 2.275137   1.86980938]
```

6.7 pickle MODULE

6.7.1 IMPORT pickle

[94]:
```
# IMPORT pickle
import pickle as pkl
```

6.7.2 dump()

[95]:
```
# dump()
sample_pkl_path = '/content/gdrive/MyDrive/Colab Notebooks/sample_pkl.pkl'
lst = [3, 5, 0, 8]
with open(sample_pkl_path, 'wb') as f:
    # 'wb' = write byte, see more below, in Chapter 7: FILES
    pkl.dump(lst, f)
```

6.7.3 load()

[96]:
```
# load()
with open(sample_pkl_path, 'rb') as f:
    # 'rb' = read byte, see more below, in Chapter 7: FILES
    loaded_content = pkl.load(f)
print(loaded_content)
```

```
[3, 5, 0, 8]
```

7 CHAPTER 7: FILES

7.1 OPENING A FILE

[97]:
```
# OPENING A FILE
f = open(hello_path, 'r')
# 'r' reading | 'w' writing | 'a' appending | 'r+' both reading and writing
```

7.2 LOOP OVER A FILE

[98]:
```
# LOOP OVER A FILE
for line in f:
    print(line, end = '')
f.close()
```

```
if __name__ == '__main__':
    print('Hello?')
```

7.3 READ WHOLE FILE

[99]:
```
# READ WHOLE FILE
f = open(hello_path, 'r')
print(f.read())
f.close()
```

```
if __name__ == '__main__':
    print('Hello?')
```

7.4 with BLOCK

[100]:
```
# with BLOCK
with open(hello_path, 'r') as f:
    print(f.read())
print(f.closed)
```

```
if __name__ == '__main__':
    print('Hello?')
True
```

7.5 WRITING FILES

[101]:
```
# WRITING FILES
write_path = '/content/gdrive/MyDrive/Colab Notebooks/writeout.txt'
with open(write_path, 'w') as f:
    f.write('Write this line to the file')
with open(write_path, 'r') as f:
    print(f.read())
```

```
Write this line to the file
```

7.6 CURRENT POSITION

[102]:
```
# CURRENT POSITION
with open(write_path, 'r') as f:
    print(f.tell())
    print(f.read())
    print(f.tell())
```

```
0
Write this line to the file
27
```

7.7 CHANGE POSITION

[103]:
```
# CHANGE POSITION
# https://www.tutorialspoint.com/python/file_seek.htm
with open(write_path, 'r') as f:
    print(f.tell())
    f.seek(13)
    print(f.tell())
    print(f.read())
    print(f.tell())
```

```
0
13
ne to the file
27
```

8 CHAPTER 8: OBJECT-ORIENTED PROGRAMMING (OOP)

8.1 CREATE A NEW OBJECT EXAMPLE

[104]:
```
# CREATE A NEW OBJECT EXAMPLE
#     name of class (parent class(es))
class Complex_number(object):
    # __init__ always run right after calling the class
    def __init__(self, a, b):
        # real and imaginary are attributes
        self.real = a
        self.imaginary = b

    # a method is a function of the class
    def modulus(self):
        from math import sqrt
        return sqrt(self.real ** 2 + self.imaginary ** 2)

    # __str__ is a method which returns (str(an instance))
    def __str__(self):
        return 'Complex number (%s + %si)' % (self.real, self.imaginary)

    # __add__ is a method which returns (an instance + another instance)
    def __add__(self, other):
        return Complex_number(self.real + other.real, self.imaginary + other.
↪imaginary)

    # __sub__ is a method which returns (an instance - another instance)
    def __sub__(self, other):
        return Complex_number(self.real - other.real, self.imaginary - other.
↪imaginary)

    # __eq__ is a method which returns (an instance == another instance)
    def __eq__(self, other):
        return self.real == other.real and self.imaginary == other.imaginary
```

The most common used methods for a class, in short:

```
__doc__: docstring  |  __name__: name  |  __del__: delete
__lt__: "<"  |  __le__: "<="  |  __ne__: "!="
__gt__: ">"  |  __ge__: ">="
__dir__: dir(an object or an instance)
```

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__matmul__(self, other)
object.__truediv__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__divmod__(self, other)
object.__pow__(self, other[, modulo])
object.__lshift__(self, other)
object.__rshift__(self, other)
object.__and__(self, other)¶
object.__xor__(self, other)
object.__or__(self, other)
```

There are so many more methods for a class that you should find them out yourself, this is the documentation of data model in Python:

https://docs.python.org/3/reference/datamodel.html

I found that the @abstractmethod decorator is not really important for the final exam, since we didn't learn any thing about decorator in Python, so if you are curious enough, this is the documentation of abc:

https://docs.python.org/3/library/abc.html

8.1.1 USING THE OBJECT EXAMPLES

[105]:
```
# USING THE OBJECT EXAMPLES
z1 = Complex_number(6, 9)
print(z1.real, z1.imaginary)
print(z1.modulus())
print(z1)
```

```
6 9
10.816653826391969
Complex number (6 + 9i)
```

[106]:
```
# USING THE OBJECT EXAMPLES
z2 = Complex_number(2.8, 9.2)
print(z2.real, z2.imaginary)
print(z2.modulus())
print(z2)
```

```
2.8 9.2
9.616652224137045
Complex number (2.8 + 9.2i)
```

[107]:
```
# USING THE OBJECT EXAMPLES
print(z1 + z2)
print(z1 - z2)
# What? Why it is -0.1999999999999993? Find out at:
# https://stackoverflow.com/questions/588004/is-floating-point-math-broken
```

```
Complex number (8.8 + 18.2i)
Complex number (3.2 + -0.1999999999999993i)
```

[108]:
```
# USING THE OBJECT EXAMPLES
print(z1 == z2)
z3 = Complex_number(6, 9)
print(z1 == z3)
```

```
False
True
```

8.2 SUBCLASS

[109]:
```
# SUBCLASS
# You can re-define any method of Complex_number, or
# add new methods, new attributes in Imaginary_number
class Imaginary_number(Complex_number):
    def __init__(self, *args):
        if len(args) == 1:
            self.removed_real_part = None
            imaginary_part = args[0]
        else: # len(args) == 2
```

```
            self.removed_real_part = args[0]
            imaginary_part = args[1]

        # 2 ways to initialize:
        # super().__init__(0, imaginary_part)
        Complex_number.__init__(self, 0, imaginary_part)

    def __str__(self):
        return 'Imaginary number (%si)' % (self.imaginary)
```

8.2.1 USING THE SUBCLASS EXAMPLES

[110]:
```
# USING THE SUBCLASS EXAMPLES
img1 = Imaginary_number(3, 8)
print(img1)
print('Removed real part:', img1.removed_real_part)
print('Modulus:', img1.modulus())
```

```
Imaginary number (8i)
Removed real part: 3
Modulus: 8.0
```

[111]:
```
# USING THE SUBCLASS EXAMPLES
img2 = Imaginary_number(4)
print(img2)
print('Removed real part:', img2.removed_real_part)
print('Modulus:', img2.modulus())
```

```
Imaginary number (4i)
Removed real part: None
Modulus: 4.0
```

9 CHAPTER 9: TESTING, DEBUGGING, EXCEPTIONS AND ASSERTIONS

9.1 TESTING AND DEBUGGING

By now, you have done this every time an error occurs in your code, so this is pretty both too easy and too hard to be explained in this notebook. Therefore, I will not explain about it here.

9.2 EXCEPTIONS

9.2.1 COMMON BUILT-IN EXCEPTIONS

[112]:
```
# COMMON BUILT-IN EXCEPTIONS
lst = [0, 1, 2]
```

```
print(lst[4])
IndexError: list index out of range
```

```
print(int(lst))
TypeError: int() argument must be a string, a bytes-like object or a number, not 'list'

print(an_undefined_a_variable)
NameError: name 'an_undefined_a_variable' is not defined

print(a

    print(a
          ^
SyntaxError: unexpected EOF while parsing

print(lst.mnpq)
AttributeError: 'list' object has no attribute 'mnpq'

print(6 / 0)
ZeroDivisionError: division by zero

print(int('?'))
ValueError: invalid literal for int() with base 10: '?'

f = open('????')
FileNotFoundError: [Errno 2] No such file or directory: '????'
```

9.2.2 (INPUT) HANDLING SPECIFIC EXCEPTIONS

[113]:
```
# HANDLING SPECIFIC EXCEPTIONS
try:
    a = float(input('a = '))
    b = float(input('b = '))
    res = a / b

except ValueError: # Handling a specific exception in try clause
    print('Unable to convert to number')
except ZeroDivisionError:
    print('Unable to divide by 0')

except: # Handling any other exception in try clause
    print('Another error')

else: # Only execute if the try clause does not raise any exception
    print(res)

finally: # Always execute
    print('Done')
```

```
a = 4
```

```
b = 8
0.5
Done
```

```
a = sfgopsgspg
Unable to convert to number
Done

a = 4
b = 0
Unable to divide by 0
Done

a = 4
b = 8
0.5
Done
```

9.2.3 raise AN EXCEPTION

```
raise ValueError('This is a value exception')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-116-d1990690242c> in <module>()
----> 1 raise ValueError('This is a value exception')

ValueError: This is a value exception
```

9.2.4 (INPUT) DEFINE AN EXCEPTION

[114]:
```
# DEFINE AN EXCEPTION
class DateFormatError(Exception):
    pass

# Raise the exception if the string is not of the format '??-??-????'
# ? is a number, from 0-9

def num_args_check(date):
    dmy_lst = date.split('-')
    if len(dmy_lst) == 3:
        return True
    return False

def numeric_check(date):
    dmy_lst = date.split('-')
    for arg in dmy_lst:
        if not arg.isnumeric():
```

```python
            return False
    return True

def args_len_check(date):
    dmy_lst = date.split('-')
    len_lst = [len(dmy_lst[i]) for i in range(3)]
    if tuple(len_lst) == (2, 2, 4):
        return True
    return False

date = input('Date = ')

if not num_args_check(date):
    raise DateFormatError(
            'There must be 3 arguments for day-month-year, respectively')
if not numeric_check(date):
    raise DateFormatError(
            'At least one of 3 arguments is not numeric')
if not args_len_check(date):
    raise DateFormatError(
            'len() of 3 arguments must be 2, 2, 4, respectively')

print('The date is: %s' % date)
```

```
Date = 66-99-6969
The date is: 66-99-6969
```

```
Date = 41-14-6161-3
DateFormatError: There must be 3 arguments for day-month-year, respectively

Date = 1-12-4251
DateFormatError: len() of 3 arguments must be 2, 2, 4, respectively

Date = 30-12-200x
DateFormatError: At least one of 3 arguments is not numeric

Date = 66-99-6969
The date is: 66-99-6969
```

9.2.5 `unittest` MODULE

A module to test your program (in a big project).

(I believe) this one will not appear in the final test, this is the documentation, though:

https://docs.python.org/3/library/unittest.html

10 APPENDIX: SOME IMPORTANT MODULES FOR MATHEMATICS, DATA SCIENCE AND MACHINE LEARNING

10.1 random

[115]:
```python
# random
# Documentation: https://docs.python.org/3/library/random.html
import random
```

10.1.1 INTEGERS

[116]:
```python
# INTEGERS
print(random.randrange(10))
# a random number in range(10)
print(random.randrange(22, 29, 2))
# a random number in range(22, 29, 2), which are 22, 24, 26 and 28
```

```
1
22
```

10.1.2 SEQUENCE

[117]:
```python
# SEQUENCE
seq = [3, 5, 0, 8]
print(random.choice(seq))
# choose a random element in a sequence
print(random.choices(seq, k = 2))
# choose a random element twice in a sequence
```

```
5
[8, 5]
```

[118]:
```python
# SEQUENCE
print(random.sample(seq, 2))
# choose two random elements in a sequence
print(random.sample(seq, len(seq)))
# alias of random.shuffle, since it is...
# Deprecated since version 3.9, will be removed in version 3.11
```

```
[5, 0]
[3, 0, 8, 5]
```

10.1.3 REAL NUMBERS

[119]:
```python
# REAL NUMBERS
print(random.random())
# a random number in the interval [0.0, 1.0)
print(random.uniform(3.0, 5.0))
# a random number between 3.0 and 5.0
# the end-point might or might not be included,
# read more in documentation
```

```
0.15902495606438138
3.504131258751076
```

10.2 time

[120]:
```
# time
# Documentation: https://docs.python.org/3/library/time.html
import time
```

[121]:
```
# time
print('before')
time.sleep(0.5)
# suspend execution for the given number of seconds
print('after')
```

```
before
after
```

[122]:
```
# time
print(time.time())
# Return the time in seconds since the epoch as a floating point number
```

```
1628488096.9642045
```

[123]:
```
# time
# measure execution time
time_before = time.time()
print(time_before)

for i in range(5):
    print(i, end = ' ')
time_after = time.time()

print('\n%s' % time_after)

print(time_after - time_before)
```

```
1628488096.9813864
0 1 2 3 4
1628488096.9855316
0.004145145416259766
```

10.3 math

```
[124]: # math
       # Documentation: https://docs.python.org/3/library/math.html
       import math
```

10.3.1 CONSTANTS

```
[125]: # CONSTANTS
       print(math.pi)
       print(math.e)
       print(math.inf)
       print(math.nan)
```

```
3.141592653589793
2.718281828459045
inf
nan
```

10.3.2 NUMBER-THEORETIC AND REPRESENTATION FUNCTIONS

```
[126]: # NUMBER-THEORETIC AND REPRESENTATION FUNCTIONS
       print(math.ceil(3.01))
       # the smallest integer greater than or equal to given number
       print(math.floor(8.99))
       # the largest integer less than or equal to given number
```

```
4
8
```

```
[127]: # NUMBER-THEORETIC AND REPRESENTATION FUNCTIONS
       print(math.gcd(15, 20))
```

```
5
```

```
[128]: # NUMBER-THEORETIC AND REPRESENTATION FUNCTIONS
       print(.1 + .2)
       print(.1 + .2 == .3)
       math.isclose(.1 + .2, .3)
```

```
0.30000000000000004
False
```

```
[128]: True
```

```
[129]: # NUMBER-THEORETIC AND REPRESENTATION FUNCTIONS
       print(math.isfinite(999999999999999))
       print(math.isfinite(math.inf))
       print(math.isfinite(math.nan))
       # nan is not finite
```

```
print()

print(math.isinf(999999999999999))
print(math.isinf(math.inf))
print(math.isinf(math.nan))
# nan is not infinite
print()

print(math.isnan(999999999999999))
print(math.isnan(math.inf))
print(math.isnan(math.nan))
```

```
True
False
False

False
True
False

False
False
True
```

[130]:
```
# NUMBER-THEORETIC AND REPRESENTATION FUNCTIONS
print(math.trunc(3.9))
print(math.trunc(3.1))
# truncated to an integral
```

```
3
3
```

10.3.3 POWER AND LOGARITHMIC FUNCTIONS

[131]:
```
# POWER AND LOGARITHMIC FUNCTIONS
print(math.exp(3)) # e^3
print(math.e ** 3)
# more accurate
```

```
20.085536923187668
20.085536923187664
```

[132]:
```
# POWER AND LOGARITHMIC FUNCTIONS
print(math.log(3))
# natural logarithm of 3
print()

print(math.log(5, 2))
# logarithm of 5 to the base 2
```

```
print(math.log2(5))
# more accurate
print()

print(math.log10(101))
```

```
1.0986122886681098

2.321928094887362
2.321928094887362

2.0043213737826426
```

[133]:
```
# POWER AND LOGARITHMIC FUNCTIONS
print(math.pow(2.1, 5.0))
print(2.1 ** 5.0)
# more accurate
print()

print(math.sqrt(5.6))
print(5.6 ** 0.5)
# more accurate
```

```
40.84101000000001
40.84101000000001

2.3664319132398464
2.3664319132398464
```

10.3.4 TRIGONOMETRIC FUNCTIONS

[134]:
```
# TRIGONOMETRIC FUNCTIONS
# all angle units are radians
print(math.sin(math.pi / 2))
print(math.cos(math.pi / 2))
print(math.tan(math.pi / 2))
print()

print(math.acos(1.0))
# arc cosine
print(math.asin(1.0))
print(math.atan(1.0))
```

```
1.0
6.123233995736766e-17
1.633123935319537e+16

0.0
1.5707963267948966
```

```
0.7853981633974483
```

10.3.5 ANGULAR CONVERSION

[135]:
```
# ANGULAR CONVERSION
print(math.degrees(math.pi / 2))
print(math.radians(90))
```

```
90.0
1.5707963267948966
```

10.4 numpy

[136]:
```
# numpy
# Documentation: https://numpy.org/doc/
import numpy as np
```

10.4.1 ARRAY CREATION

[137]:
```
# ARRAY CREATION
print(np.array( [1, 4, 6, 9] ))
print(np.array( ((1, 3), (6, 9)) ))
print(np.array( [(3.2, 2.1), (6.0, 1.3)] ))
# a list/tuple of lists/tuples of... is allowed
print()

print(np.array( [6, 9, 3, 0], dtype = float ))
print(np.array( [(3.2, 2.1), (6.7, 1.3)], dtype = np.int8 ))
# there are a lot of data types available,
# check it out in the documentation
```

```
[1 4 6 9]
[[1 3]
 [6 9]]
[[3.2 2.1]
 [6.  1.3]]

[6. 9. 3. 0.]
[[3 2]
 [6 1]]
```

[138]:
```
# ARRAY CREATION
print(np.arange(10))
print(np.arange(21.2, 8.1, -2.5))
# floats can be passed to arange
print()

print(np.linspace(2.3, 4.8, 5))
```

```
# arrays with a specified number of elements,
# and spaced equally between the specified beginning and end values
```

```
[0 1 2 3 4 5 6 7 8 9]
[21.2 18.7 16.2 13.7 11.2  8.7]

[2.3   2.925 3.55  4.175 4.8  ]
```

[139]:
```
# ARRAY CREATION
# most types of special two-dimensional matrices in linear algebra
# can be created using np functions, find out more in the documentation
print(np.eye(3))
print(np.diag([3, 4, 5]))
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[3 0 0]
 [0 4 0]
 [0 0 5]]
```

[140]:
```
# ARRAY CREATION
print(np.zeros((2, 3)))
print()
print(np.ones((3, 2, 4)))
```

```
[[0. 0. 0.]
 [0. 0. 0.]]

[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
```

10.4.2 INDEXING

[141]:
```
# INDEXING
np_arr = np.arange(3.3, 12, 1.1)
print(np_arr)
print(np_arr[-2])
print(np_arr[-2:])
# slicing np array is exactly the same as slicing python list
```

```
[ 3.3  4.4  5.5  6.6  7.7  8.8  9.9 11. ]
9.900000000000002
```

```
[ 9.9 11. ]
```

[142]:
```
# INDEXING
print(np_arr.shape)
np_arr.shape = (2, 4)
print(np_arr.shape)
print(np_arr)
```

```
(8,)
(2, 4)
[[ 3.3  4.4  5.5  6.6]
 [ 7.7  8.8  9.9 11. ]]
```

[143]:
```
# INDEXING
print(np_arr[1])
print(np_arr[1, 2])
```

```
[ 7.7  8.8  9.9 11. ]
9.900000000000002
```

[144]:
```
# INDEXING
print(np.arange(4))
np_arr_2 = np.arange(1, 8, 2)
print(np_arr_2)
print()

index_arr = np.array([1, -1, 3])
print(index_arr)
print(np_arr_2[index_arr])
# a np array can be used to select element
print()

index_arr_2 = np.array( [[1, 0], [3, 2]] )
print(index_arr_2)
print(np_arr_2[index_arr_2])
print()
# indexing np arrays which have more dimensions is more complicated,
# read more in documentation
```

```
[0 1 2 3]
[1 3 5 7]

[ 1 -1  3]
[3 7 7]

[[1 0]
 [3 2]]
[[3 1]
 [7 5]]
```

[145]:
```
# INDEXING
np_arr_3 = np.arange(9)
print(np_arr_3)
index_arr_3 = np_arr_3 % 2 == 0
print(index_arr_3)
print(np_arr_3[index_arr_3])
# in data science, filtering data is very important,
# so this is a must-have tool
```

```
[0 1 2 3 4 5 6 7 8]
[ True False  True False  True False  True False  True]
[0 2 4 6 8]
```

10.4.3 BROADCASTING

[146]:
```
# BROADCASTING
# the term broadcasting describes how np treats arrays
# with different shapes during arithmetic operations
np_arr_a = np.array([6, 8, 9])
print(np_arr_a)
np_arr_b = np.ones(3, dtype = np.int8)
print(np_arr_b)
print(np_arr_a + np_arr_b)
```

```
[6 8 9]
[1 1 1]
[ 7  9 10]
```

[147]:
```
# BROADCASTING
print(np_arr_a + 1)
# read more about general broadcasting rules for
# more-dimensional-arrays in the documentation
```

```
[ 7  9 10]
```

10.4.4 STRUCTURED ARRAYS

[148]:
```
# STRUCTURED ARRAYS
# there are a lot of concepts for this type of array,
# the following is only a simple example
struct_arr = np.array([('Alan', 19, 169.2), ('Paul', 21, 172.3)],
                      dtype = [('name', 'U10'), ('age', 'i4'), ('height', 'f4')]
                      )
print(struct_arr)
```

```
[('Alan', 19, 169.2) ('Paul', 21, 172.3)]
```

10.5 MOST USED MODULES FOR MATHEMATICS, DATA SCIENCE AND MACHINE LEARNING

- TensorFlow: https://www.tensorflow.org/
- SciPy: https://www.scipy.org/
- Matplotlib: https://matplotlib.org/
- Pandas: https://pandas.pydata.org/
- Keras: https://keras.io/
- scikit-learn: https://scikit-learn.org/
- statsmodels: https://www.statsmodels.org/
- Plotly: https://plotly.com/
- Seaborn: https://seaborn.pydata.org/