PYTHON

DATE 10/12/2023

→ **Introduction :**

- Guido Van Rossum Invented Python in 1990
- Python 3 was released in 2008.
- Python is easy to use language, designed for clear logical coding.
- Lots of Existing libraries in Python.

  Ex. Numpy, Scipy, SciKit-Learn, Pandas, Matplotlib, Plotly, Seaborn, PySpark

→ **Uses :**

- Automate Simple Tasks

  i.e. Work with PDF's, Excels, text messages, etc
- DS and ML

  i.e. Create Visualisations, Analyse large data files, etc

→ **Command Line Basics :**

(i) ⊞ + R

(ii) Type 'cmd' to open Command Prompt

→ Commands :

cd : Returns Current Directory

pwd : Returns Working Directory

dir : Returns Contents of the Current Directory

cd "NAME" : Change Location of Current Directory into a folder existing in the main directory

cd.. : Go back by one directory

cls : Clears the entire screen of the command prompt

[TIP : Press TAB to autocomplete a folder's Name]

→ **Installation :**

Anaconda Distribution

- → Includes not only Python but other libraries as well
- → Own Virtual Environment System
- → 'All-In-One' Install for DS and ML

Installing :

(i) Check all tickboxes [i.e. Add to PATH & REGISTER]

(ii) Install to Default Directory

DATE

Then, Open Anaconda Navigator

↳ Jupyter Notebook

Jupyter Notebook does not require network connectivity.

↳ They are of the form '.ipynb' meaning i python notebooks & Can only be opened on Jupyter Notebooks.

Opening Jupyter Notebook easily:

(i) Open cmd and type 'jupyter notebook'.
It will open a notebook in the cmd location specified.

(ii) Enter "localhost8888' in the search bar of chrome.

TIPS:

(1) Shift + ↵ in a notebook executes a code a goes to the below cell.

(2) Alt + ↵ in a notebook executes a code and creates a new cell below.

(3) Stopping a python code, ex. Infinite loop being run, can be done by restarting Kernel.

(4) Code option can be changed to markdown cell and vice-versa from the home menu bar. Useful to save notes.

→ Running Code:

(1) Install Subline-Text.
Code and save it as a .py file.
To run it, Open a cmd and enter
'python filename.py'.

(2) Enter 'python' in a cmd window.
Only Single Lined codes can be written.
Use 'quit()' to exit.

→ Free "NO-INSTALL" Options:

- jupyter.org/try
- Google Collab Online Notebooks
- Repl.it

- Google Search - "Python Interpreter Online"

Disadvantages:

- Hard to upload own code, data or import notebooks.
- May not save code in free versions
- May change from free to paid or might get shut down, etc.

→ Environments:

↳ Actual Thing in which Python code will be typed in.

3 Main types of Environments:

(1) Text Editors

- General Editor for any text file
- Work with variety of file types, not exclusively for python.

  Ex. HTML, JavaScript, Python, etc
- Customisable with Plugins & Add-onns

  Ex. Sublime Text, Atom

(2) FULL IDEs

- Specifically designed for Python
- Larger sized programs will be handled more conveniently.
- Only community editions are free.

  Ex. PyCharm, Spyder, etc

(3) Notebook Environment:

- Great for Learning, User-friendly [i.e. Beginner-Friendly]
- Supports In-Line Notes, Visualisations, Videos
- Not .py extension files.

  Ex. Jupyter Notebooks - .ipynb

→ Virtual Environments:

↳ Allow you to setup virtual Installations of Python and libraries onto your computer.

- You can have multiple versions of Python or libraries & can easily activate or deactivate these environments

Ex. Want to program in different versions of a library, Making sure library Installations are in correct locations, etc

DATE 12/12/2023

→ Virtualenv library for normal Python

• But Anaconda has a built in Virtual Environment manager.

Steps:

(1) Open cmd and type:

'conda create --name NAME LIBRARYNAME

(2) To activate, type

'activate NAME'

To deactivate that virtual environment named NAME,

'deactivate NAME'

(3) If you want to install other libraries, type

'conda install LIBRARYNAME

[Ex.

conda create --name mypython3 python = 3.5 numpy]

(4) List all Virtual Environments, type

'conda info --envs'

→ Topics Covered:

- Data Types:

  Numbers, Strings, Lists, Dictionaries, Tuples, Sets, Booleans and Print formatting
- Comparison Operators
- if, elif and else statements
- Loops
- range()
- List Comprehensions
- Functions
- Lambda Expressions
- Map & filter

13/12/23

→ Data types:

① Numbers → Integer, Float (Decimal)

↳ Numerical Information

Other than 'int' and 'float', there also exists 'complex'

Every Program should start with:

#!/usr/bin/env python3

DATE 13/12/2023

→ Operators:

+ ↔ Addition

- ↔ Subtraction

/ ↔ Division, // → Division (int) [0.1+0.2-0.3 ≠ 0.0]

* ↔ Multiplication

% ↔ Remainder

** ↔ Power

Priority: () > * > + (or) -

→ Precedence of Operators in descending order: ←

( )	Left to Right
**	Right to left
+x, -x, ~x	Left to Right
*, /, //, %	Left to Right
+, -	Left to Right
>>, <<	Left to Right
&	Left to Right
^	Left to Right
|	Left to Right (Bitwise OR)
!=, <, <=, >, >=, ==	Left to Right
Not x	Left to Right
And	Left to Right
Or	Left to Right
If else	Left to Right
Lambda	Left to Right
=, +=, -=, *=, /=	Right to left

→ Assigning variable names:

Example: x = 2

Rules for ~~assigning~~ variable names:

(1) Names cannot start with a number. (Starts with _ or letter)

(2) No spaces can be used in a name (Instead use _)

(3) Can't use :'," <>/?|\()!@#$%*~-+ symbols

Variable starting with underscore in python are known as private variables.

dunder variables should be left alone, i.e. Variables starting with two underscores. No variable should be assigned as dunder variable conventionally.

DATE

(4) Can't be keywords ; i.e.,

and	del	from	nonlocal	True
as	elif	global	None	try
assert	else	if	not	with
break	except	import	or	while
class	finally	in	pass	yield
continue	false	is	raise	
def	for	lambda	return	

(5) Avoid using datatypes as names, For ex. list, str, etc

→ Python uses dynamic typing

i.e. reassign variables to different data types

Ex. cat = 2, bat = 3 (OR) cat, bat = 3, 2

cat = "Sam"

C, C++ use static typing, Hence produces error.

Advantages :

- Easy to work with
- Faster Development time

Disadvantages :

- May result in bugs
- type() checks type of variable

Ex. a = 10 → Used to comment

a = a+a # a = 20 now

type(a) # Output : int

If a → 10.0, type(a) returns float.

(2) Strings :

Ordered Sequences of characters i.e. they are indexed

Ex. a = "Hello", 'hello', 'hi how are you'

H	e	l	l	o
0	1	2	3	4
0	-4	-3	-2	-1

} (Indexing)

bin(number) → returns binary form (base 2) of the given arguement (number)

bin(5) → returns 0b101

Expression : Part of code which produces a value, i.e without "="

Statement : Line of code performing action, i.e. with "="

mystring.startswith("ch") → Starting check DATE

Slicing allows you to grab a subsection of multiple characters.

Indexing allows you to grab a particular element.

↳ i.e. [start : stop : step]

start : numerical index for the slice start

stop : numerical index for the slice end [not included]

step : size of the jump

Single quote : 'hi I am Harith'

Double quote : "hi I'm Harith"

print("hello world") ⟹ hello world

print("hello \nworld") ⟹ hello
world

print("hello \tworld") ⇒ hello world

Note : len('I am') ⟹ 4 [Spaces are also counted] ←

↳ Used to find out length of string

→ Slicing / Indexing :

(i) Ex. my_string = "I am good"

print(mystring[3]) ⟹ m

print(mystring[-2]) ⇒ o

(ii) Ex. my_string = "hello I am Good"

print(my_string[2:]) ⇒ llo I am Good

print(my_string[:3]) ⇒ hel

print(my_string[4:9]) ⇒ o I a

print(my_string[::2]) ⇒ hloI od ←

print(my_string[2:13:3]) ⇒ l aG

Trick :

print(my_string[::-1]) ⇒ dooG ma I olleh

Alternative :

print("hello I am Good"[::-1])

Typecasting binary into integer base 10 :

int("0b101", 2) ⟶ returns 5

↳ form in which python stores binary numbers

Long strings can be created using 3 single quotation marks. Can be in multiple lines

long-string-ex = """hello
i am fine"""

DATE

→ Strings are Immutable, i.e. Can't use indexing to change individual elements in a string

Ex. name = "Sam"

name[0] = 'p' ← Error

But name.replace('a', 'b') returns 'Sbm'.

Third argument can be no. of occurances to be replaced

Instead:

Ex. 'p' + name[1:]

Ex. x = "Hi"

x = x + "How are you?"

Output: Hi How are you?

Ex. x = "Hi"

print(x * 10)

Output: HiHiHiHiHiHiHiHiHiHi

Note: (i) 2 + 3 = 5

'2' + '3' = 23

(ii) Strings cannot be added to numbers. Error.

→ String functions:

x = 'Hi How are you'

x.upper() → converts the string to uppercase

x.lower() → converts the string to lowercase

x.split() → splits the string into lists based on the white spaces (Default: White space)

x.capitalize() → Capitalizes only first letter of string.

x.upper() ⇒ HI HOW ARE YOU

x.lower() ⇒ hi how are you

x.split() ⇒ [Hi, How, are, you]

x.split('o') ⇒ [Hi H, w are y, u]

x.capitalize() ⇒ 'Hi how are you'

→ Print formatting with string:

Ex. myname = 'Hari'

print("My name is" + myname)

AKA string Interpolation

Two ways:

(i) .format() method

(ii) f-strings method

x.strip() → returns a string without any spaces at the end or beginning of the given string

classmate PAGE

x.find(substring) → Used to find starting Index of a substring

x.title() → Returns the string but first letter of every word capitalized

Methods are like functions but attached to the datatypes of the object.

DATE

(i) Ex. : print ('This is a string {}'.format('INSERTED'))

Output : This is a string INSERTED

Ex : print ('A {2} {1} {0}'.format ('good', 'lives', 'boy'))

Output : A boy lives good.

good → 0th Index in .format()

lives → 1st Index in .format()

boy → 2nd Index in .format()

Ex. print ('A {1} {1}'.format ('good', 'bad', 'sad'))

Output : A sad sad

Ex. print ('A {b} {l} {g}'.format (g='good', l='lives', b='boy'))

Output : A boy lives good

Note : This is not only used for string.

Ex. res = 1/3

print ("The result is {}".format(res))

↳ The result is 0.333333333

Float Formatting Syntax :

{value : width.precision f} → Automatically Rounds off

Ex. print("The result is {:1.3f}".format(res))

Output : The result is 0.333

Ex. print("The result is {r:5.3f}.format(r=res))

Output: The result is 0.333.

(ii) Ex. : print(f'Hello, this string {res}')

If res = 'NAME',

Output : Hello, this string NAME

Ex. name = "Hari"

age = "18" (or) age = 18

print (f"My name is {name} and I'm {age} years old.")

Output : My name is Hari and I'm 18 years old.

classmate Opposite of string split() function, there exists PAGE 9

''.join(list) function which joins the list of entries with space in between them here as an example.

dir(datatype) returns a list of all the methods that can be applied to that datatype. Ex. dir(str)
They will be there after the double underscore methods.

DATE

③ Lists :

- Ordered Sequences of objects that can hold variety of object types
- Supports Indexing and Slicing

Ex. emylist = [1, 2, 3]
mylist = ['abc', 1, 20.3]

Note : len(mylist) returns the no. of elements in the list

Ex. mylist[0] → abc
mylist[1:] → 1, 20.3
[0]*3 → [0, 0, 0]

Note : Adding two lists is possible, i.e. concatenation
Ex. emylist + mylist returns
[1, 2, 3, 'abc', 1, 20.3]
Ex. newlist = emylist + mylist

→ Lists can be mutated unlike Strings.
i.e. newlist[1] = 'HELLO'
Now newlist returns [1, 'HELLO', 3, 'abc', 1, 20.3]

NOTE :

(i) newlist.append('help') appends / concatenates 'help' at the end of the list. It permenantly changes the list

[TIP : Press TAB after newlist. to get options / functions]

(ii) newlist.pop() removes the last entry from the list and returns it.
(OR) newlist.pop(index) removes the given index, [Default index is -1.] and returns it.

(iii) newlist.sort() does not return anything. It sorts the list contents into alphabetical order in case of list having alphabet strings & increasing order in case of list having numbers.

classmate

import builtins
dir(builtins)
↳ Shows all the available functions (built-in ones) in Python

PAGE

a, b, *other, c = [1, 2, 3, 4, 5]

↳ a = 1
b = 2
other = 3, 4 → Takes rest of them
c = 5

DATE 14/12/2023

IMP :

mysortedlist = ~~newlist~~ newlist.sort()

is pointless as there is no returning for sorting function.

Printing mysortedlist would result empty.

∵ type(mysortedlist) = NoneType

∴ another list cannot be assigned to sorted list this way

To Rectify this,

newlist.sort()

mysortedlist = newlist

(iv) newlist.reverse() reverses all elements in the list.

(v) sorted(newlist) returns the sorted list as a list.

→ Nested Lists :

Ex. my_list = [1, 1, [2, 3]]

print(my_list[2][1]) ⟹ Output : 3

∵ my_list[2] → returns [2, 3]

Ex. lists = [1, 2, [6, 9, ['target', 'hello']]]

print(lists[2][2][1]) ⇒ Output : hello

14/12/23

④ Dictionaries :

- Unordered mappings for storing objects
- Key Value pair, No Need of Indexing
- Syntax :

{'key1': 'value1', 'key2': 'value2'}

Dictionaries	Lists
(i) Objects retrieved by key name	Objects retrieved by location
(ii) Unordered and can't be sorted	Ordered and can be indexed

[Key has to be unique]

- Ex. prod = {'key1': 'Apple', 'key2': 'Orange'}

prod['key1'] ⇒ Apple

PAGE 11

classmate

Keys must be immutable objects, Ex. strings, booleans, Integer but not List.

Conventionally a string.

d.get(key) returns None if that key is not present in the dictionary

d.clear() clears the dictionary

d2 = d.copy() creates a copy DATE

d.popitem() pops the last key value pair

- Dictionaries can store integers, floats and even lists.
- Nested dictionaries:

  d = {'k1': 123, 'k2': 12.05, 'k3': [1,2,ab], 'k4': {kk: 100}}

  d['k4']['kk'] ⟹ 100

Ex. d = {'k1': [a, 2, c]}

d[k1][2].upper() ⟹ C

- Adding entries to the dictionary

  d = {'k1': [a, 0.2, 3], 'k2': 'app'}

  d['k3'] = 300

- Changing value of a particular entry

  d['k1'] = 'Apple' ∴ Dictionaries are Mutable.

Note:

(i) d.keys() returns all the keys in form of Tuple

(ii) d.values() returns all the values in form of a Tuple

(iii) d.items() returns all the pairings in form of a Tuple

[Note: Keys should always be strings]

d.get(key, def. value) returns def. value if key given doesn't exist

⑤ Tuples:

- Similar to List but Tuples are Immutable.

  i.e. Once inserted, an entry cannot be reassigned.

- Paranthesis: ( )

  Ex. (1,2,3)

- t = (1, 4, 6, 9)

  len(t) returns the no. of elements in the tuple [Here, 4]

- Slicing and Indexing works
- Different datatypes can be inserted as well.

Note:

(i) t.count('name') returns the no. of times the string 'name' exists in the tuple

(ii) t.index('name') returns the index of the 'name' string

Methods of a set :

s. discard(element)	s1.isdisjoint(s2)	intersection : &
s1.difference(s2)	s1.issubset(s2)	Union : |
s1.intersection(s2)	s1.issuperset(s2)	
s1.union(s2)		

DATE

only the index where it first exists, if there are multiple.

⑥ Sets :

- Unordered collection of unique elements
  i.e. only one representative of the same object
- Paranthesis : { }
- Ex. myset = set() [Basically appends at the end of set]
  myset.add(1) ⟹ myset → {1}
  myset.add(2) ⟹ myset → {1,2}
  myset.add(e) ⟹ myset → {1,2}

- listmy = [1,1,6,6,6,6,7,7,7,3,3,3,3,3,3,3]
  set(listmy) ⟶ {1,6,7,3}
- {1,1,6,6,6,7,7,9,9} ⟹ {1,6,7,9} [Returns unique objects]
- Sets do not have ordering
- Ex. set('hhaaarrrrii') ⟹ set → {h,a,r,i}

⑦ Booleans :

- True and False (case-sensitive)
- type(True) → bool
- Ex. 1>2 returns False
  1==1 returns True

1 is 1 returns True	
[] is [] returns False	

[Note : To define a variable for using it later in the code, use
x = None]

⟹ Input/Output with Basic files in Python : (.txt)

Note : The following only works on Jupyter Notebook.

```
%%writefile myfile.txt
Hello this a text file
Second line
Hi this third line
```

This writes a .txt file. To access it,

'is' checks if both are exactly the same, and stored in same location

nonlocal x = 10
↳ changes the global variable's value

DATE

myfile = open('myfile.txt')
If myfile.txt does not exist, file not found error is displayed (Errno 2)
But checks only in current directory.
Type pwd to get current working directory

To read a text file : myfile.read()
i.e. Displays all the content into a single string

- Python escape characters [i.e. Backslash constant]
  - \n : New line
  - \a : Audible bell
  - \b : back space
  - \' : Single Quote (Displays)
  - \" : Double Quote (Displays)
  - \t : Horizontal tab space
  - \v : Vertical tab space
  - \? : Display Question Mark
  - \0 : Null character (End of string)
  - \\ : backslash (Displays)
  - \r : Carriage Return [i.e. takes cursor to start of line]

If you want to read file again,
myfile.read() ⟹ ''
Nothing gets displayed ∵ Cursor is already at the EOF.
~~EOF : End Of file~~
[EOF : End Of file]
∴ myfile.read() ⟶ Displays
myfile.seek(0) ⟹ Returns 0 and Cursor sets to SOF
myfile.read() ⟹ Displays again

Ex. content = myfile.read()
To read line by line and save each line in a list, we use myfile.readlines(). It returns a list.
myfile.writelines() writes into a file.

classmate

PAGE

if ((n := len(a)) > 10):
    print(f"{a}")

:= Operator assigns and can be reused.

To go one directory back from the folder in which main.py file is and then go into files directory,

file = open("..\\files\\filename.txt", 'r') DATE

To open file from some other location on the PC,

myfile=open("c:\\Users\\UserName\\Folder\\test.txt")

(OR) myfile = open(r"C:\Users\UserName\Folder\test.txt", "r")

After working with the file, it is important to close it to avoid getting any errors.

myfile.close()

Note:

with open('myfile.txt') as file_variable_name:

~~contents = filem file_variable_name.open()~~

contents = file_variable_name.read()

(i) Indentation gets automatically created (default: A tabspace) of 4 spaces and whatever code written inside Indentation lies within it.

(ii) This way, No need to close file.

TIP:

While typing "with open('myfile.txt')", if you place cursor after the open bracket and press ↑ + TAB, predefined functions get shown.

Works with any function

Note:

Default mode: r

with open('myfile.txt', mode='r') as my_file:

contents = my_file.read()

This doesn't work with other modes.

Available modes: r ↔ read only

w ↔ write only

a ↔ append only

r+ ↔ reading and writing

w+ ↔ writing and reading

w mode creates a file if given file does not exist.

params, *args, default parameters, **kwargs

DATE / / /

Ex. with open('my_new_file.txt', mode='r') as f:
print(f.read()) ⇒ Now cursor is at EOF
with open('my_new_file.txt', mode='a') as f:
f.write('Fourth line') ⇒ Appends on the third line
(OR) f.write('\n Fourth line') ⇒ Creates fourth line

Ex. with open('random.txt', mode='w') as f:
f.write('New file created')

Q) Write a .py code that opens a file name test.txt and writes "Hello World" to the file and then closes it.

A) with open('test.txt', mode='w') as f:
f.write("Hello World")

[(OR)]

myfile = open('test.txt', mode='w')
myfile.write('Hello World')
myfile.close()

Q) Call 'Hello' from d = {'k1': [{'nest_key': ['this is deep', ['hello']]}]}

A) d['k1'][0]['nest_key'][1][0]

Q) 3.0 == 3 . True or False

A) True

→ Comparison Operators:
a == b : Equality
a != b : Inequality
a < b : less than
a > b : Greater than
a >= b : Greater than or equal to
a <= b : Less than or equal to

→ Logical Operators

a and b :

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

a or b :

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

not a :

a	not a
True	False
False	True

→ if, elif, else statements :

↳ control flow i.e. conditioned

Control Flow is very crucial to Python. It makes use of Colons and Indentation (whitespace)

Syntax :

① if condition :
    code

② if condition :
    statement 1
else :
    statement 2

③ if condition 1 :
    statement 1
elif condition 2 :
    statement 2
else :
    statement 3

DATE

Note : Python will only execute the first elif ~~condition~~ block which is true even if there are other true ~~ones~~ elif blocks below it.

→ FOR LOOPS :

Syntax :

```
my_iterable = [1,2,3]
for item_name in my_iterable :
    print(item_name)  ⟹ 1
                        2
                        3
```

Note :

① print(item_name) can be ~~represented~~ replaced by print('Hello'). Hello will get displayed 10 times.

② item_name plays the role of 'i'.

③ for - Else Loop also exist if there is break inside for.

Q) my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

(i) print only even numbers.

(ii) Print sum of all number

Sol: (i)

```
for i in my_list :
    if (i % 2 == 0) :
        print(i)
    else
        print(f'Odd no. : {i}')
```

(ii)

```
sum = 0
for i in my_list :
    sum = sum + i
print(f'The sum is {sum}')
```

Note :

(i)

```
for letter in 'What is my name' :
    print(letter)
```

(ii)

```
for _ in (2,4,6) :
    print('Nice')
```

classmate Just like Switch-case in C ; In Python, there is match-case.

Same syntax (No default req.) (No break after each case)

```
match variable :
    case 'a' :
        print("Hi")
```

→ For default, Use case something: (or) case _: (convention)

PAGE

all(list) returns true if all the given elements in the list given are all True.

DATE

(iii) mylist = [(2,6), (7,8), (1,7), (9,5)]

```
for item in mylist:
    print(item)
```
Prints tuple pairs

```
for (a,b) in mylist:
    print(b)
```
This is called Tuple Unpacking

```
for a,b in mylist
    print(a)
```

(iv) mydict = {'k1': 2, 'k2': 3, 'k2': 7}

```
d = mydict
for item in mydict
    print(item)
```
prints only keys

```
for item in d.items():
    print(item)
```
prints key value pairs

```
for key, value in d.items():
    print(value)
```

```
for val in d.values():
    print(val)
```
prints only values

→ **WHILE LOOP :**

Syntax :

(i) while booleaned-condition:
    statement 1
    statement 2

(ii) while condition:
    statement 1
else:
    statement 2

Note : (No Fullcolon after these)

(i) break : breaks out of the current closest enclosing loop

(ii) continue : Goes to the top of the closest enclosing loop

(iii) pass : Does not do anything

DATE

Use of pass :

```
x = [1,2,3]
for _ in x :
    # nothing here
```
gives error

```
x = [1,2,3]
for _ in x :
    pass
```
No error

Use of continue :

```
mystring = "sammy"
for letter in mystring :
    if letter == 'a' :
        continue
    print(letter)
```

Output :

s
m
m
y

Use of break :

```
mystring = 'sammy'
for letter in mystring :
    if letter == 'a'
        break
    print (letter)
```

Output :

s

→ Some special Operators :

① Range — range()

Syntax :

```
for num in range (start, end):
    print (num)
```

Note :

(i) Start value is 0 by default if not provided

(ii) End value is not included

(iii) Can even use jump

```
for num in range (start, end, jump):
    print (num)
```

(iv) Can also create the output (transform) into a list

```
list(range(0,11,2))
```

DATE

② Enumerate - enumerate()

Syntax:

```
word = 'abcdef'
for item in enumerate(word):
    print(item)
```

Output results in pairs, i.e. tuples

(0, 'a')
(1, 'b')
(2, 'c')
⋮
(5, 'f')

```
for item, charect in enumerate(word):
    print(charect)
```

③ Zip - zip()

Syntax:

```
mylist1 = [1, 2, 3]
mylist2 = ['a', 'b', 'c']
```

zip(mylist1, mylist2) ⟶ creates a zip

```
for item in zip(mylist1, mylist2):
    print(item)
```

Output: A set of tuples

(1, 'a')
(2, 'b')
(3, 'c')

list(zip(mylist1, mylist2)) ⟶ creates a list of tuples

Note:

To check if something is in a list or not,

2 in ['a', 1, 3] ⟶ False

2 in ['a', 2, 'b'] ⟶ True

Similarly for a character in a string, or a key in a dictionary, But for keys or values, use

DATE

123 in d.values()
'key2' in d.keys()

④ min, max in a list

Syntax:

mylist = [10,20,80,100,60]
min(mylist) ⟶ 10
max(mylist) ⟶ 20

⑤ random library

(i) from random import shuffle → built-in library
mylist = [1,2,3,4,5,6,7,8]
shuffle(mylist) → Shuffles the list and does not return anything (NoneType)

(ii) from random import randint
randint(0,100) → Choses and Returns a Random Integer
mynum = randint(0,10)

⑥ Input from User:

input('Enter a Number:')
This returns the number entered by the user.
mynum = input('Enter:')

Note:

(1) Input() always takes string even though a number is entered for typecasting,
float(mynum) (OR)
int(mynum)
Hence overall,
mynum = int(input('Enter:'))

⑦
```
x = [1,2,3,4]
out = []
for num in x:
    out.append(num**2)
```

(OR)

```
x = [1,2,3,4]
out = [num**2 for num in x]
```

classmate

PAGE

DATE

⑦ → List Comprehension

Ex. mystring = 'hello'

mylist = []

for item in mystring:

mylist.append(item)

(OR) mylist = [item for item in mystring]

Output : mylist → ['h', 'e', 'l', 'l', 'o']

Ex. mylist = [num for num in range(0,10)]

mylist = [num**2 for num in range(0,10)]

mylist = [num**2 for num in range(0,10) if num % 2 == 0]

Ex. celcius = [10, 20, 0, 50, 5]

fahrenheit = [(9/5)*temp + 32) for temp in celcius]

↳ Appending

↳ Creating a new list called fahrenheit.

for if-else statements in List Comprehensions,

results = [x if x % 2 == 0 else 'ODD' for x in range(0,10)]

→ Nested loop:

mylist = []

for x in [2,4,6]:

for y in [100, 200, 300]:

mylist.append(x*y)

Output : [200, 400, 600, 400, 800, 1200, 600, 1200, 1800]

By using ~~reee~~ list comprehension:

mylist = [x*y for x in [2,4,6] for y in [100, 200, 300]]

→ Ternary Operator:

<condition_if_true> if <condition> else <condition_if_false>

classmate print( ) has an 'end = ' ' ' argument which adds to newline.

PAGE 23

DATE 17/12/2023

Note:

help() returns some help regarding a particular command.

Ex. help(mylist.insert)

mylist = [1,2,3] was executed before

→ FUNCTIONS:

Syntax:

```
def name_of_function():
    """
    This function can have arguments/parameters.
    """
    Statement-1
```

Docstring

Ex.
```
def name_fun():
    print('hello')
name_fun() ⇒ hello
```

```
def name_fun(name):
    print('hello' + name)
name_fun(joe)
hello joe
```

Note:

We use return keyword to send back the result of the function instead of printing it out. It allows us to assign the output of the function to a new variable

Ex.
```
def add_fun(num1, num2):
    return num1+num2
result = add_fun(3,5)
print(result)
```

Note:

(i) When calling functions in python, Paranthesis is a must.

(ii) If Parametres are not given while calling a function defined with parameter, it will display an error

To prevent this,

```
def name-fun(name = 'Default'):
    print(f'hello {name}')
```

ISSUE:

If Inputs for add-fun are given as strings, then result will be a concatenation of the strings

Ex.
```
def even-check(num):
    return num%.2==0
```

Ex.
```
def check_even_list(num_list):
    for num in num_list:
        if num %.2==0:
            return True
        else
            pass
```

→ Acts like break too

→ "return False" cannot be written

This returns TRUE if Any number is even in a list.

Ex.
```
def check-even_list(num_list):
    for num in num_list:
        if num%.2==0:
            return True
        else
            pass
    return False
```

Note: Tuple Unpacking can also be done by functions

For example:

~~ex_tuple = [('A',10),('B',12),('C',10)]~~

~~alph, num = ex_tuple~~

```
ex_tuple = ('A', 10)
alph, num = ex_tuple
print(alph) ⟶ A
```

DATE 19 12 2023

Tip:

For error handling in input:

```
while guess not in ['0','1','2']:
    guess = input('Pick a number: 0, 1, or 2')
```

→ *args - Arguements

**kwargs - Keyword Arguements

For example:

```
def myfunc(a,b, c=0, d=0, e=0, ...):
    return sum((a,b,c,d,e,...)) * 0.05
```

*args allows us to take arbitrary number of parameters.

Ex.
```
def myfunc(*args):
    return sum(args) * 0.05
```

Here args will be a tuple of inputs (returns)

*name can be used instead of *args, but conventionally *args is chosen.

Ex.
```
def myfunc(**kwargs):
    if fruit in kwargs:
        print('My fruit of choice is {}'.format(kwargs['fruit']))
    else:
        print('I did not find any fruit here')
```

Input: myfunc(fruit = 'apple', veg = 'spinach')

↳ My fruit of choice is apple.

kwargs will be a dictionary of inputs (returns)

**name can be used instead of **kwargs, but conventionally **kwargs is chosen.

Example of Combination:

```
def myfunc(*args, **kwargs):
    print('I would like {} {}'.format(args[0], kwargs['food']))
myfunc(10,20,10, fruit='orange', food='apples')
```

↳ I would like 10 apples.

Note:

Order should be maintained.

i.e. in the above example, keyword arguments cannot be passed before arguments. — Error.

Q) Write a Python Program where you define a function myfunc that takes in a string and returns a modified string such that every even letter is lowercase and every odd letter is uppered

Sol:

```
def myfunc(string):
    newstring = ''
    for index, char in enumerate(string):
        if index % 2 == 0:
            newstring += char.lower()
        if index % 2 == 1:
            newstring += char.upper()
```

→ Note:

① .join() is used to ~~two~~ join two strings in a list with a connector string.

Ex. `" ".join(['Hi', 'How', 'are'])`
returns "Hi How are"

`"--".join(['abc', 'DEF'])`
returns "abc--DEF"

② abs(num) returns the absolute value of an integer 'num'

DATE 21 12 2023

Q) Define a function which takes arbitrary number of arguements and returns a list containing only those arguements which are even.

Sol:
```
def myfunc(*args):
    return [eveniten for eveniten in args if eveniten % 2 == 0]
```

→ map() function maps a function to multiple parametres

Ex.
```
def square(num):
    return num**2
my_nums = [1,2,3,4,5]
map(square, my_nums) → returns map location
for item in map(square, my_nums):
    print(item)
```
↳
```
1
4
9
16
25
```

If the output needs to be in a list format,

```
list(map(square, my_nums))
```
↳ [1,4,9,16,25]

i.e. returns a list

Note:

Function needs to be passed in map() as an arguement because map() will execute the function itself.

→ filter() function returns a memory location where it stores only True valued outputs.

Ex.
```
def check_even(num):
    return num % 2 == 0
mynums = [1,2,3,4,5,6]
list(filter(check_even, mynums))
```
↳ [2,4,6]

abs(num) → Displays absolute value of num

round(num) → Rounds num to nearest Integer