**PYTHON ESSENTIALS LAB GUIDE**

This document serves as a guide to practice what has been taught by the instructor in the training session. The course participants can use this as a helper guide to practice the below, during the **student lab session** of the training. When the student practices the below material, it acts as a re-enforcer of the concepts learnt during the training.

## **Day 1: Python in a Nutshell**

**Lab: Using Python Shell**

Task 1: Running hello world program : Running as script / program / comments

>>> print(‘hello world’)

$ python3 1\_hello.py (or)

$ chmod +x 1\_hello.py ;

$ ./1\_hello.py

Task 2: Using Python shell

>>> 6000 + 4523.50 + 134.25

10657.75

>>> \_ + 8192.75

18850.5

>>>

Task 3: Defining Variables

>>>42 # int

>>>4.2 # float

>>>'forty-two' # str

>>>True # bool

>>>x = 42

>>> s = ‘Hello world’

>>> type(x)

>>>type(s)

Task 4: Writing a program on shell with variables

Type in 2\_interest.py

# interest.py

>>>principal = 1000 # Initial amount

>>>rate = 0.05 # Induce Error by introducing indentation error

>>>numyears = 5 # Number of years

>>>year = 1

>>>while year <= numyears:

principal = principal \* (1 + rate)

print(year, principal)

year += 1

>>> print(f'{year:>3d} {principal:>0.2f}')

6 1276.28

>>> print(f'{year + 10 :>3d}')

16

>>> print(f'year: {year + 10 :>3d}')

year: 16

Task 5: Performing Arithmetic operators

>>> 7 / 4

1.75

>>> 7 // 4

1

>>> 7 % 4

3

For floating-point numbers, the modulo operator returns the floating-point remainder of x // y, which is x – (x // y) \* y.

>>> 3.5 % 2

1.5

>>> round(0.5) #Round prefers an even number nearest to the input

0

>>> round(1.5)

2

>>> round(3.5)

4

>>> pow( 2, 3 )

8

>>> pow( 2, 3 , 6) # (2 ^ 3) % 6

2

Task 6: Performing Bitmask operations

>>> a = 0b11001001

>>> mask = 0b11110000

>>> x = ( a & mask ) >> 4

>>> x

12

Task 7: Logical operations

>>> y = 0

>>> x and y

0

>>> y = None

>>> x and y

>>> x

12

>>> y

>>> res = x and y

>>> res

>>>

Task 8: Storing the result after doing a computation

>>> x = x + 1

>>> x

13

>>> x += 1

>>> x

14

**Text Strings**

Task 15a: Defining string variables

>>>a = 'Hello World'

>>>b = "Python is groovy"

>>>c = '''Computer says no.'''

>>>d = """Computer still says no."""

Task 15b: Print variables with f ‘string’ & as a sequence of variables

Task 16: Printing multi-line strings

>>>print('''Content-type: text/html

>>><h1> Hello World </h1>

>>>Click <a href="http://www.python.org">here</a>.

>>>''')

Task 17: Simple String Processing

a = 'Hello World'

print(len(a)) # 11

b = a[4] # b = 'o'

c = a[-1] # c = 'd'

Task 18: String Indexing & Concatenation

g = a + 'ly' # g = 'Hello Worldly'

c = a[:5] # c = 'Hello'

x = '37'

y = '42'

z = x + y # z = '3742' (String Concatenation)

z = int(x) + int(y) # z = 79 (Integer Addition)

Task 19: Defining lists : Take example from the PPT

names = [] # An empty list

names = list() # An empty list

Task 20: Defining tuples:

holding1 = ('TATAMOTORS', 540, 294.1)

address = ('www.python.org', 80)

a = () # 0-tuple (empty tuple)

b = (5,) # 1-tuple (note the trailing comma)

Task 21: Defining sets . Sets can contain immutable objects as its elements

names1 = { 'IBM', 'MSFT', 'AA' }

names2 = set(['IBM', 'MSFT', 'HPE', 'IBM', 'CAT'])

>>names2

>>names[2] ## What does it give

Task 22: Create a dictionary

s = {

'name' : 'GOOG',

'shares' : 100,

'price' : 490.10

}

>>>company\_name = s[‘name’] #Prints ‘GOOG’

>>>cost = s['shares'] \* s['price']

Task 23: Simple conditionals

>>>a = 10

>>>b = 6

>>>if a < b:

>>> print('Computer says Yes')

>>>else:

print('Computer says No')

>>>if a < b:

pass # Do nothing

>>>else:

print('Computer says No')

Task 24: Multiple if conditionals

>>>suffix = ‘.jpg’

>>>if suffix == '.htm':

content = 'text/html'

>>>elif suffix == '.jpg':

content = 'image/jpeg'

>>>elif suffix == '.png':

content = 'image/png'

>>>else:

raise RuntimeError(f'Unknown content type {suffix}')

Task 25: Simple while loop Prints numbers 1 through 9

>>>x = 0

>>>while (x := x + 1) < 10: # Prints 1, 2, 3, ..., 9 ; x:=x + 1 walrus operator

print(x)

Task 26 : For loop

>>> x = 0

>>>for x in range(10):

print(x)

Task 26 : Usage of break statement. Same as above but stops when x is 5 . **Ask the participants to do this at their end**

**Explain the syntax: for : else:**

>>>x = 0

>>>for x in range(10):

>>> if x == 5:

break # Stop the loop. Moves to Done below

>>> print(x)

>>> x += 1

>>> print('Done')

Task 27: Usage of continue Same as above but continues when x is 5 , hence skips printing 5

>>>x = 0

>>>while x < 10:

>>> x += 1

>>> if x == 5:

>>> continue # Skips the print(x). Goes back to loop start.

>>> print(x)

>>>print('Done')

Task 28: Iterating list elements

>>> names[2] = ['Kumar', 10, 45]

>>> names

for name in names:

print(name)

**File Input and Output**

Task 29: Getting input from keyboard

name = input('Enter your name : ')

print('Hello', name)

Task 30 : File handling

with open('data.txt') as file:

for line in file:

print(line)

#print(line, end='') # end='' omits the extra newline

**Functions:**

Task 31: Defining a function & invoke it

def remainder(a, b):

'''

Computes the remainder of dividing a by b

'''

q = a // b

r = a - q \* b

return r

>>> remainder(8,6)

**Ask the participants to write a function add\_2\_num() to return sum of 2 numbers**

**Exceptions:**

Task 32: Unhandled exceptions crashes the program

* 1. >>> s = 'he'
  2. >>> i = int(s)
  3. Traceback (most recent call last):
  4. File "<stdin>", line 1, in <module>
  5. ValueError: invalid literal for int() with base 10: 'he'
  6. >>>

Task 33: Exceptions if handled gracefully terminates the program

>>> s = 'he'

>>> try:

... i = int(s)

... except ValueError as err:

... print(f's={s}')

... print(err)

...

s=he

invalid literal for int() with base 10: 'he'

>>>

**Switch to AWS Python setup**

**Task : Ask the participants to write a program interest.py for calculating compound interest**

principal = 1000 ; rate = 5% ; years = 5 ; print the principal after 5 years

Demo: Extend it ask from the user. Do it as a demo..

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Basic \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

## **Sequence & Collection Data Types**

**Strings - Detail:**

Task 40: Printing& Escaping Characters in Multi-line statements

Run str\_1.py

Task 41: Indexing a String

>>>flavour = ‘fig pie’

>>>flavour[3] ??

>>>flavour[10] ???

>>>final\_index = len(user\_input) - 1

>>>last\_character = user\_input[final\_index]

>>>flavour[-1]

Run the program str\_2.py (Indexing)

Task 42: String Concatenation **(Ask the participants whats the output . Earlier covered topic)**

>>>a = ‘Hello World’

>>>g = a + 'ly' # g = 'Hello Worldly' (String concatenation)

>>>string1 = "Navin"

>>> string2 = "Kumar"

>>> magic\_string = string1 + string2

>>> magic\_string

Task 43: Slicing a String

>>>a = ‘Hello World’

>>>c = a[:5] # c = 'Hello'

>>>d = a[6:] # d = 'World'

>>>e = a[3:8] # e = 'lo Wo'

>>>f = a[-5:] # f = 'World'

>>>flavour = ‘fig pie’

>>>len(flavour)

>>>flavour[:3] ???

>>>flavour[13:15] # Does Not complain if its out of range

>>>flavour[15] #Throws an exception

>>>flavour[-7:-4] ???

Run the program str\_3.py (Slicing )

Task 44: Slicing & Striding together

Run the program str\_4.py (Slicing )

Task 45: String Methods

>>>’Navin Kumar Gopalakrishnan’.lower()

>>>name = ‘Navin Kumar Gopalakrishnan’

>>>name.lower()

>>> name = ‘Navin Kumar GopalaKrishnan ’

>>>name.rstrip()

>>>name.lstrip()

>>> name.rstrip()

>>> starship = ‘Enterprise’

>>> starship.startswith("en") #False

>>>> starship.startswith("En")

Run the program str\_5.py (String methods)

Task 41: Repr representation of a string

s = 'The value of x is ' + str(x)

s = 'The value of x is ' + repr(x)

>>> s = 'hello\nworld'

>>> print(str(s))

hello

world

>>> print(repr(s))

'hello\nworld'

>>> print(f'The value of string s is:{s!r}') # Guess what would be the answer

>>> print(f'The value of string s is:{s}')

Task 46: Iteration of a sequence (iterable)

>>>for ch in g:

>>> print(ch, end=’ ’)

>>> #print(ch, end=’ ’)

## Explain the mechanism : raise StopException by for loop

Task 47: Repetition operator

>>> s3 = '=' \* 5

print(s3)

print(len(s3))

>>> num = ‘12’

>>> num \* 3

>>> 3 \* num

>>> ‘12’ \* ‘3’ ??? Functions as a multiplication operator

**Lists (Intermediate)**

Task 50: Changing List elements

>>> names[2] = ['Kumar', 10, 45]

>>> names

['Navin', 'Pravin', ['Kumar', 10, 45], 'Nikhil', 'Sita', 'Arjun']

names = [ 'Navin', 'Pravin', 'Nathalie','Nikhil', 'Sita' ,'Arjun']

Task 51: List Concatenation

a = ['x','y'] + ['z','z','y'] # Result is ['x','y','z','z','y']

**Tuples (Intermediate)**

Task 52: Common usage of tuples is to unpack the elements

>>holding1 = ('TATAMOTORS', 540, 294.1)

>>address = ('www.python.org', 80)

name, shares, price = holding1

Task 53: Combining lists & tuples

holding2 = ('EXIDE', 250, 167.5)

portfolio = [holding1, holding2]

total = 0.0

for name, shares, price in portfolio:

total += shares \* price

**Sets (Intermediate)**

Task 55: Defining sets . Sets can contain immutable objects as its elements **(Ask the participants)**

names1 = { 'IBM', 'MSFT', 'AA' }

names2 = set(['IBM', 'MSFT', 'HPE', 'IBM', 'CAT'])

>>names2

>>names[2]  **## What does it give???**

Task 57: Unordered property of sets (Do for 1st time.Ask the participants to repeat it 2 times after closing the Python shell Enter again )

>>>for item in names1:

print(item)

Repeat it .

Repeat it closing the interpretor and opening a new Python shell

**Ask what's your Observation**

\*\*\*\*\*\*\*\*\* FINISHED UNTIL THIS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Dictionaries (Intermediate)**

Task 56 : Create a dictionary . Inserting keys in dict

s = {

'name' : 'GOOG',

'shares' : 100,

'price' : 490.10

}

>>>company\_name = s[‘name’] #Prints ‘GOOG’

>>>cost = s['shares'] \* s['price']

>> s[‘shares’] = 300

>>>s[‘date’] = ‘2020-03-25’

>>>s

Task 57: Another Dict example. **Ask the participants to do**

prices = {

'GOOG' : 490.1,

'AAPL' : 123.5,

'IBM' : 91.5,

'MSFT' : 52.13

}

**\*\*\*\*\*\*\*\*\* START HERE FOR TODAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Lists / Sets / Tuples / Dictionaries (Detail)**

Task 60: Initializing a list

Run list1.py

Task 61: is operator

>>> a = [1, 2, 3]

>>> b = [1, 2, 3]

>>> a is b

False

>>> a == b

True

>>>

Task 62: Membership operator

>>> 9 in [1,2,3,6,7]

Task 63: Functions available on list

Run list3.py

Task 64: List methods

dir(list) # List all methods for the data type list

Run

Task 67: List manipulation

Run list\_manipulation.py

**Run list\_manipulation\_2.py**

Task 68: List splicing & striding

Run list\_splicing.py

Task 69: List comprehension

Run list8.py

Run list9.py

**Ask the participants to do list4.py using list comprehension**

Task 70:

>>>prices = {

'GOOG' : 490.1,

'AAPL' : 123.5,

'IBM' : 91.5,

'MSFT' : 52.13

}

Task 71: Access by indexing with keys

>>> prices[‘GOOG’]

Task 72 : Updation of an item

>> prices[‘GOOG’] = 600

>> prices

Task 73: Membership operator

>>> 'GOOG' in prices.keys()

True

>>> 'GOOG' in prices

True

>>> 'TATAMOTORS' in prices

False

>>> 'TATAMOTORS' in prices.keys()

False

Task 74: Alternate ways of creating a dict

Run dict1.py with d = dict(pairs)

Task 75: Nested Dictionaries (Creating nested dictionaries in slide)

Run dict2.py

Task 76: Dictionary Comprehensions

Run dict\_comprehension.py

Task 77 : Dictionary methods

>>>if 'IBM' in prices:

p = prices['IBM']

else:

p = 0.0

Task 78: Accessing and setting a default value if the key doesn’t exist

>>>> p = prices.get('IBM', 0.0) # prices['IBM'] if it exists, else 0.0

>>>prices[('IBM', '2015-02-03')] = 100.5 #Using tuples as Multi-key in a dict

**Stop & Switch to slide “Dictionaries vs lists”**

**Set – Detail (Ignore)**

Task : Operations on sets

Run set1.py

a | b gives different results when the same operation is done after closing the Py shell. Ie sets are unordered collection data types ie no notion of a sequence

**Conditional / Comparison / Logical Operation (Detail )**

Task 90: Ternary Operator

>>>maxval = a if a > b else b

>>>if a > b:

maxval = a

>>>else:

maxval = b

Task 91: Comparison operator

>>> "apple" < "astronaut"

>>> "beauty" > "truth"

>>> "good" != "bad"

Task 92 : Logical Operators

>>> 1 < 2 and 3 < 4 # Both are True

>>> 2 < 1 and 3 < 4 # First statement is False

>>> 1 < 2 or 3 < 4 # Both are True

>>> 1 < 2 or 4 < 3 # Second statement is False

>>> x = 0

>>> not x

>>> ("A" != "A") or not (2 >= 3)

Task 93 : Conditions using if / else /elif

>>>if 2 + 2 == 5:

print("Is this the mirror universe?")

Task 94 : if-else:

Run 1\_condition.py

Task 95: if/elif/else

Run 2\_condition.py

Task 96 : Nested ifs

Run 3\_condition.py

Task 98: Usage of if with for loops

Run 5\_condition.py

Task 100: Password alert using for and else loop **(Ask the participants to write a function Alert\_passwd()**

* Take password , count from user
* Alert\_password should take it as arguments and check whether the password is the same as : I<3Bieber
* If successful come out of the loop. For every unsuccessful attempt the line “Password is incorrect” should be printed. After the count is exceeded, it should print the line ‘Suspicious activity. The authorities have been alerted.’

Run 8\_condition.py

Task 101: Nested ifs with logical operators & if/elif/else conditions (Ignore)

Run 4\_condition.py

\*\*\*\*\*\*\*\*\*\*\*\* End of Day 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## **Functions**

**Functions**

Task 1: Whats a function

>>> len

<built-in function len>

>>> type(len)

<class 'builtin\_function\_or\_method'>

>>> len = "I'm not the len you're looking for."

>>> len

"I'm not the len you're looking for."

>>> type(len)

<class 'str'>

>>> len() #Raises error

>>> num\_letters = len("four")

Run remainder.py

Explain : Function Body & Function Signature

i) Having only one definition

ii) Both definitions

iii) With print('Where am I') outside the remainder()

iv) With print('Where am I') inside the remainder()

v) Move the print(remainder()) at the top of the file

Task 2 : No return values

Run no\_return.py

Task 3 : Compound interest

Run 2\_compound\_interest.py

Task 4: Comparing 2 objects

Run 3\_function.py

Task 5: String Processing

Run stringprocess.py

Task 6: Functions are names stored in procedural table

* Run if\_func.py on Python Shell
* Change the value to debug = 0 . The same old definition of square() is present in memory. To change it , again define square() on the Python Shell

Task 7: Positional argument

Run area\_triangle.py

Show by typing only partial area\_ in VSCode press Tab

Error: Only 2 arguments are passed

Task 8: Positional argument or keyword argument

>>> string.ascii\_letters

'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

>>>

Run keywordarg.py

Task 9: Variable Arg

Run variable\_arg\_to\_func.py

Task 10 : Positional & key word args together

Run pos\_and\_keywordargs.py

Task 11: Restricting positional argument

Run star\_pos.py

Task 12: Keyword arguments

Run keyword1.py

Task: Both variable & keyword arguments

Run both\_var\_and\_keyword.py

Task 13: Scope Resolution

>>> x = 2

>>> x

2

>>> x = 3

>>> x

3

Task 14 : LEGB

Run scope\_func1.py , Run scope\_func2.py , scope\_func3.py

Task 15: Simple lambda function on Shell

Run lambda1.py

Task 16: More lambda functions

Run lambda2.py

Task 17: More formal way of writing functions

Run shorten.py

**Exceptions:**

Task 1: Overview of Exceptions

Run 1\_exception.py

Task 2: Writing try except blocks

Run 2\_exception.py . Show them while typing div... then Press **Tab** it completes

Task 3: Portfolio Example

Run 2\_readport.py

Task 4: List\_find example of

Run list\_find.py

**Show the slide Exception Hierarchy**

Task 5: Exception Hierarchy

Run exception\_hierarchy.py

Task 6: Generate ValueError

Run age.py with different definitions of get\_int()

Task 7: Generate IOSError Exception:

Run noblanks.py

At the end of the program ask what happens when an image file is passed as input?

**Show the slide Flow-Diagram-Exception Handling**

Traceback (most recent call last):

File "noblanks.py", line 21, in <module>

lines = read\_data(filename)

File "noblanks.py", line 10, in read\_data

for line in fh:

File "/usr/lib/python3.8/codecs.py", line 322, in decode

(result, consumed) = self.\_buffer\_decode(data, self.errors, final)

UnicodeDecodeError: 'utf-8' codec can't decode byte 0x89 in position 0: invalid start byte

**File Input and Output (in Detail)**

**File Handling – Detail**

Task 1: Using Path library

# echo -n 'Hello world' > hello.txt

>>> from pathlib import Path

>>> path = Path.cwd() / 'hello.txt'

>>>with path.open(mode = 'r', encoding = 'utf-8') as fh:

data = fh.read()

>>> data

<\_io.TextIOWrapper name='/home/navin/hello.txt' mode='r' encoding='utf-8'>

>>>> fh

>>>> data = fh.read() # ValueError: I/O operation on closed file.

Task 3: Read & Write text files

Run file3.py

Task 4: Write multiple lines

Run file4.py

Task 5: Append Mode

Run file5.py

Task 6: File Operations

Run Os\_Sys\_Modules/ 3\_file\_operations.py

Task 7: Directory Operations

Run Os\_Sys\_Modules/ 4\_dir\_operations.py

Task 8: Sys modules

Run Os\_Sys\_Modules/pcost.py

## **Modules**

**Modules**

Task 1: Explain whats a module

Run simple.py

a) Run as script: python3 simple.py

b) Use Python shell: import simple . Loading of the module happens only once. 1st time its cached.

Only once its run, the next time import statement is executed it will not be run, because its cached. The better way to reload the changed module is

to come out of the shell and re-load the module.

c) Isolation / namespace of the variables in a module:

- from simple import x, run

- When spam() , it says spam is not defined...

- x is pinned to the file simple.py (alternatively to the module simple)

- x = 13, run() is executed still prints

d) In shell: type \_\_name\_\_ / simple.\_\_name\_\_ to see the values

Task 2: Practical explanation of module eg. readport.py , port.py

- Run readport.py in the shell by importing

- Put if \_\_name\_\_ in readport.py

Task 3: Different import statements import as rp, from readport import \*

Task 4: **Give as an exercise** Modify code to implement a list comprehension to calculate the value of the portfolio

total = sum([shares \* price in for name, shares, price in my\_portfolio])

Task 6: Using CSV Modules to write

Run file6.py

Task 7: Read CSV File

Run file7.py

Task 8: Write a CSV File with headers

Run file9.py

Task 9: Read a CSV File with headers

Run file8.py

Task 65: **Give this as assignment for file handling, list , string**

Run list4.py

Task 1: Deep Copy

Run : 3\_module.py

Task 5: Run Task 4 with portfolio\_bad.csv

Task 4: Can use noblanks.py to show import sys to process command line arguments

functionality

Run the original noblanks.py file

## **Object Oriented Programming**

* 1. **OOP:**

Task 1 : OOP Primer

Explain Class & Object with examples from slides (Ganesh Chathurthi, Villa Model etc)

Task 2: Point Class

Doing on Python Shell:

>>> import Shape

>>> a = Shape.Point()

>>> a

Point(0, 0)

>>>type(a)

>>> b = Shape.Point(3,4)

Explain whats meant by repr() and str(b) . Go to

Task 41: Repr representation of a string

>>> s = 'hello\nworld'

>>> print(str(s))

hello

world

>>> print(repr(s))

'hello\nworld'

s = 'The value of x is ' + str(x)

s = 'The value of x is ' + repr(x)

>>> print(f'The value of string s is:{s!r}') # Guess what would be the answer

>>> print(f'The value of string s is:{s}')

>>> str(b)

'(3, 4)'

>>> b.distance\_from\_origin()

5.0

>>> b.x = -19

>>> str(b)

'(-19, 4)'

>>> a == b , a != b

(False, True)

>>>

Explain \_\_new\_\_ -> \_\_init\_\_ -> returning a reference and stored in the variable self

Run Shape.py as : python3 Shape.py

Task 3: Circle Class inherits from Point Class

Run Shape.py with Circle class on Python Shell

>>> import Shape

>>> p = Shape.Point(28,45)

>>> q = Shape.Point(28,45)

>>> c = Shape.Circle(5,28,45)

>>> p.distance\_from\_origin()

53.0

>>> c.distance\_from\_origin()

53.0

>>>

Task 4: Str, int, float are all classes

>>> s = "Hello World"

>>> t = ‘Navin’

>> s + t

>> s == t

Task 5: Account Class

Run Account1.py

>>>import Account1

>>> a = Account1.Account(‘Navin’,1000.0)

>>> type(a)

<class 'Account'>

>>> type(b)

<class 'Account'>

>>> type(a).deposit

<function Account.deposit at 0x10a032158>

>>> type(a).inquiry

<function Account.inquiry at 0x10a032268>

>>>

Task 6: Attribute access

>>> a = Account('Guido', 1000.0)

>>> a.owner # get

'Guido'

>>> a.balance = 750.0 # set

>>> del a.balance # delete

>>> a.balance

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'Account' object has no attribute 'balance'

>>>

Task 7: Creation of Attributes

Attribute operations: Getting, Setting, Deleting

>>> a = Account('Guido', 1000.0)

>>> a.creation\_date = '2019-02-14'

>>> a.nickname = 'Navs’

>>> a.creation\_date

'2019-02-14'

Task 8: Using getattr(), setattr(), delattr(), hasattr()

>>> a = Account('Guido', 1000.0)

>>> getattr(a, 'owner')

'Guido'

>>> setattr(a, 'balance', 750.0)

>>> delattr(a, 'balance')

>>> hasattr(a, 'balance')

False

>>> getattr(a, 'withdraw')(100) # Method call

>>> a

Account('Guido', 650.0)

>>>

Task 9: Bound Method

>>> a = Account('Guido', 1000.0)

>>> w = a.withdraw

>>> w

<bound method Account.withdraw of Account('Guido', 1000.0)>

>>> w(100)

>>> a

Account('Guido', 900.0)

>>>

A bound method is an object that contains both an instance (the self) and the function that implements the method. When you call a bound method by adding parentheses and arguments, it executes the method, passing the attached instance as the first argument. For example, calling w(100) above turns into a call to Account.withdraw(a, 100).

Task 10: Scoping Rules

Run Account2.py

Although classes define an isolated namespace for the methods, that namespace does not serve as a scope for resolving names used inside methods. Therefore, when you’re implementing a class, references to attributes and methods must be fully qualified.

>>> from Account2 import Account

>>> a = Account('Guido', 1000.0)

>>> a.withdraw(100)

>>> a

Account('Guido', 900.0)

Task 11: Operator Overloading and Protocols – Portfolio of Accounts

Run Account3.py

Sometimes you’ll hear the word “Pythonic,” as in “this code is Pythonic.” The term is informal, but it usually refers to whether or not an object plays nicely with the rest of the Python environment. This implies supporting—to the extent that it makes sense—core Python features such as iteration, indexing, and other operations. You almost always do this by having your class implement predefined special methods

Task 12: Inheritance – adding a new method

Run Account3.py

Task 13: Re-defining an existing method inquiry

Run Account4.py

Run this code multiple times to see the effect of re-defined inquiry() method

Task 14: Using super() method

Run Account4.py commenting : No super() definition of class EvilAccount

Run this code multiple times to see the effect of re-defined inquiry() method

Task 15: Creating new Attributes in the instance of a child class

Run Account5.py :

Run this code multiple times to see the effect of re-defined inquiry() method

Task 16: Using the right class when a class is a subclass of another class

Run Account6.py

Change \_\_repr\_\_() in Account1.py

Re-Run Account6.py

Task 17: Parent & ChildClass relationship checking

Run with correct \_\_repr\_\_() in Account1.py

>>> from Account6 import \*

>>> a = EvilAccount('Navin', 1000.0)

>>> type(a)

<class 'Account6.EvilAccount'>

>>> isinstance(a,Account)

True

>>> isinstance(a,object)

True

Task 18: Inheritance problem

Run Stack1.py

It has methods of lists also.. like sorting(), reverse() those are not required for a Stack

Task 22: Class variables

Run classvar1.py