PYTHON SUMMARY

Herut Sterman

Basic List Operations

Expression	Result	Description
arr=[1,2,3,4], len(arr)	4	Length
[6 ,5 ,4] + [3 ,2 ,1]	[6 ,5 ,4 ,3 ,2 ,1]	Concatenation
s="hello", s2=s*2	hellohello	Repetition
4 in arr	true	Membership
print(max(arr))	4	maximum

Built-in List Functions & Methods

arr.count(obj) - Returns count of how many times obj occurs in arr

arr.index(obj) - Returns the lowest index in list that obj appears in arr

arr.insert(index, obj) - Inserts object obj into list at offset index

arr.remove(obj) – Removes the first occurrence of obj in arr

arr.reverse() - Reverses objects of list in place

arr.sort() - Sorts objects of list, use compare func if given

Python – Dictionary

The values in dictionary items can be of any data type:

String, int, boolean, and list data types:

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
print(thisdict) → {"brand": "Ford", "electric": False, "year": 1964, "colors":
["red", "white", "blue"]}
```

Python Tuples

Updating Tuples

```
tup1 = (12, “a”)
tup2 = 1, 3, 6, “tuple”

tup3 = tup1 + tup2 → (12, “a”, 1, 3, 6, “tuple”)
tup4 = tup1*2 → (12, “a”, 12, “a”)
```

Membership

```
12 in tup1 →  true
12 not in tup1 →  false
```

Iteration

```
for x in tup2:
    print (x, end=“ ”)
```

Output: 1 3 6 “tuple”

Classes

```
class Point:
    “”” this class represent a 2d point in the plane “””
    num = 0
    #  initilizes the point according to its coordinates: (x, y)
    #  the default values: x=0, y=0
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        Point.num = Point.num + 1

    # returns a String contains the Point data

    def __str__(self):
        return “[“ + str(self.x) + “,” + str(self.y) + “]”

    # returns the distance from the point to the origin

    def dist(self):
        return math.sqrt(self.x**2 + self.y**2)
```

Copy objects

אנו משתמשים ב-**copy** module של Python לצורך פעולות העתקה רדודות ועמוקות.

נניח, שיש להעתיק את רשימת מורכבת בשם x:

```
import copy
scopy = copy.copy(x)
dcopy = copy.deepcopy(x)
```

Inheritance

```
# file A.py
class A:
    def __init__(self, x=0):
        self.x = x
        print("A init")

    def method(self):
        print("parent_method")

    def get_attr(self):

        print("x = ", self.x)

    def set_attr(self, x):
        self.x = x

# file B.py
from A import A
class B(A):
    def __init__(self, x):
        super().__init__(x)
        print("B init")

    def method(self):
        print("child_method x = ", self.x)
```

```
# file Test.py

from A import A
from B import B

a = A(3)
b = B(99)
b.get_attr()
b.set_attr(55)
b.get_attr()
a.method()
b.method()
```

Output:

```
A init
A init
B init
x = 99
x = 55
parent_method
child_method x = 55
```

Interface

Informal Interfaces ממשקים לא פורמליים

בנסיבות מסוימות, ייתכן שלא נזדקק לכללים המחמירים של ממשק פייתון רשמי. האופי הדינמי של פייתון מאפשר לנו ליישם ממשק לא פורמלי. ממשק לא רשמי הוא מחלקה המגדירה שיטות שניתן לדרוס אותן, אך אין אכיפה קפדנית.

לצורך הדוגמא ניקח מחלקה אבסטרקטית שמייצגת צורה ומכילה שתי פונקציות אבסטרקטיות לחישוב שטח והיקף של צורה:

```
class Shape:
    def area(self) -> float:
        pass

    def perimeter(self) -> float:
        pass
```

Unitest

דוגמה: בדיקת פונקציה המקבלת מספר שלם ומחזירה אמת אם המספר הוא ראשוני:

```
import unittest
from testtests import MyFunctions

class MyTestCase(unittest.TestCase):

    def test_is_prime1(self):
        b = MyFunctions.is_prime(2)
        self.assertEqual(b, True)

    def test_is_prime2(self):
        b = MyFunctions.is_prime(8)
        self.assertEqual(b, False)


if __name__ == '__main__':
    unittest.main()
```

Output:

```
Ran 2 tests in 0.038s
OK
```

Writing to a File

```
f=open("myfile.txt","w")
f.write("Hello! I love Python”)
f.close()
```

- The f=open("myfile.txt","w") statement opens myfile.txt in write mode.
- The open() method returns the file object and assigns it to a variable f.
- "w" specifies that the file should be writable.
- This statement stores a string in the file.
- In the end, f.close() closes the file object.

Reading from a File

```
f=open("myfile.txt","r")
line=f.readline()
while line!='':
    print(line)
    line=f.readline()
f.close()
```

Use the for loop to read a file easily:

```
f=open("myfile.txt","r")
for line in f:
    print(line)
f.close()
```

with operator

אופרטור with ב - Python משמש בטיפול בחריגים כדי להפוך את הקוד לנקי וקריא הרבה יותר.

זה מפשט את ניהול המשאבים הנפוצים כמו זרמי קבצים. שימו לב לדוגמא הקוד הבאה כיצד השימוש ב- with משפט הופך את הקוד לנקי יותר.

```
# without using with statement
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
    file.close()

# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

שימו לב שבניגוד לישום הראשון, אין צורך לקרוא לפונקציה **file.close ()** בעת שימוש אופרטור **with**. שימוש באופרטור **with** מבטיח שחרור נכון של המשאבים.

Exceptions

```
arr = [10, 0, 2]
for entry in arr:
        try:
            rand = 1/int(entry)
            print("rand = ", rand)
        except Exception as e:
            print(sys.exc_info()[0], " occurred.")
Output:
rand = 0.1
<class 'ZeroDivisionError'> occurred
rand = 0.5
```

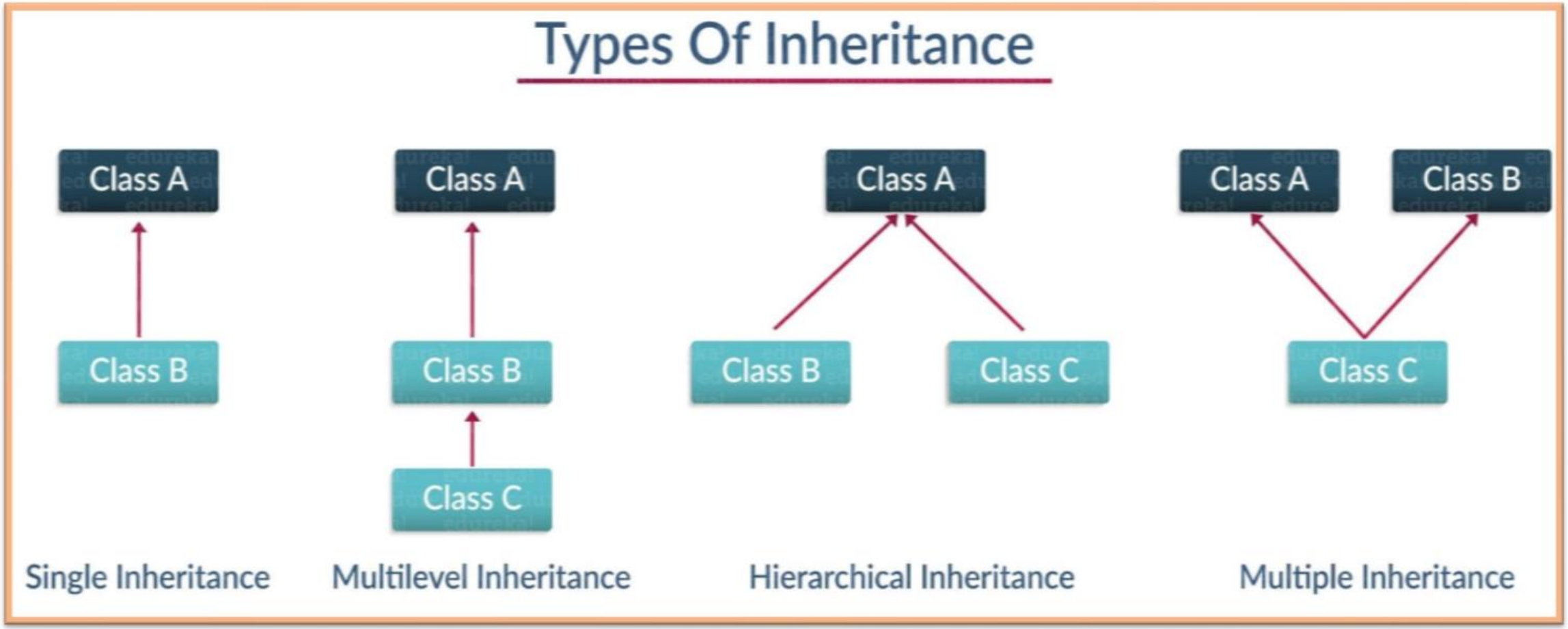
Logging

```
import logging
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
logging.basicConfig(filename='app.log', filemode='w',
                    format='%(name)s - %(levelname)s - %(message)s')
logging.warning('This will get logged to a file’)
```

OUTPUT ON CONSOLE

```
WARNING:root:This is a warning message
ERROR:root:This is an error message
CRITICAL:root:This is a critical message
WARNING:root:This will get logged to a file
```

Types Of Inheritance



Lambda Expressions

```
double = lambda x: x* 2
print(double(4))

-------------------------


 #Regular function
def add(x, y):
    return x + y

 #Lambda expression
add = lambda x, y: x + y
```

GUI

- GUIs are composed of components (e.g., buttons, labels) that can be represented as objects.
- Each GUI component has:
  - State (e.g., button text, color)
  - Behavior (e.g., on-click actions)
- OOP principles applied:
  - Encapsulation: GUI components manage their own state and behavior.
  - Inheritance: Base classes for generic components; derived classes for specialized ones.
  - Polymorphism: Different components can respond differently to the same event.

Pandas

Introduction to pandas Data Structures

pandas has two main data structures it uses, namely, *Series* and *DataFrames*.

pandas Series

pandas Series one-dimensional labeled array.

pandas DataFrame

pandas DataFrame is a 2-dimensional labeled data structure.

Pandas - useful operations

- index
- loc, iloc
- pop, del
- insert
- read_csv
- head, tail
- describe
- groupby
- shape
- isnull, dropna
- show, plot
- value_counts, mean
- sort_values
- merge
- to_datetime