Installing python

Python + visual studio code on windows / mac linux.

Python shell.

If you dont apply a file to the python command. python will open in ondemand mode. This is known as python interaction shell console. This accepts python directly. good for quick testing. But the code you write inside python shell cannot be saved.

Run a python file

by pressing ▷ button. or

```
py -3.9 file name. py        (windows)
python -3.9 file name. py

python 3 filename. py   (ubuntu)
```

Starting python shell.

```
Python 3     ubuntu linux mac
py -3        windows.
```

Declaration

```
x = 10 (int) implicit methode
x = "10" (string, character)
```

Lists

```
listname = [   ]
```

we can automatically create a list of number using range

```
l = list( range(1,10))        [1, 2.... 9]
l = list( range(1, 10, 2)
```

specify a step as 3rd argument

[1, 3, 5 .7, 9]

Attributes

dir(list) shows methods / attributes
dir(__builtins__) shows builtin functions
help() shows function properties

lists - append() - append an object to list
clear() - clear the list
index() - shows the index

print(listname[1:4] print index 1 to 3
[:2] 0 to 1
[3:] 3 to last item
[-1] prints last items
python has 2 indexing positive & negat
print(listname[-4:-2] prints -4 -3 inde

Accessing & slicing strings

mystring = 'hello'
print(mystring[1]) = 'e'
monday = ["hello", 1, 2, 3]
print(monday[0]) = "hello"
Print(monday[0][2]) = 'l'

Dictionary

Items in dictionary are made up of keys and values.

dictionary = {"name": 10, "a": 10.3}

Print(dictionary.values()) print values
Print(dictionary.keys()) print keys

Print (dictionary ["key"])

Touple

Immutable

In a list you can add values with append i.e, lists are dyanamic. But in the case of touple there are no append or remove methods they are faster than lists

```
touple      = (....)
List        = [.....]
dictionary  = {.....}
```

Python is mainly used for automation purpose, web apps and data science eg: facebook uses python for image processing

converting b/w data types

touple to list

```
>>> data = (1, 2, 3)
>>> list (data)        = [1, 2, 3]
```

list to touple

```
>>> data = [1, 2, 3]
>>> tuple (data)       = (1, 2, 3)
```

list to dictionary

```
>>> data = [["name", "john"], ["surname", "smith"]]
>>> dict (data)
= {"name": "john", "surname": "smith"}
```

Creating functions

```
def functionname (parameters):
    . . . .
    . . . .
    return variable
```

If the function doesnot have a re
statement it returns a none object.

conditional statements

Based on a condition the statement
executed, we can use, and, or operat
with conditional statements

```
if condition :
        statement
else ~~condition~~ :
        statement
```

```
def    mean(value):
       if type(value) == dict:
           mean = sum(value.values())/len(v
       else:
           mean = sum(value)/len(value)
   return mean
monday_temp = [5, 8, 9, 9.1, 9.9]
grades = {"mary": 9, "Anu": 9.2, "dc": 9.8}

print mean(grades)
```

isinstance() function checks if the valu
is of particular type

```
isinstance(value, dict)
```

User Input

```
Input ('Enter a number')
user_input = float(input("enter amo")
```

because by default python will assign string data type to the variable

String formatting

```
user-input = Input ("Enter name")
message = "Hello %s" % user-input
print (message)
```

or

```
message = f"Hello {user-input}"
```

python3

formatting with multiple variable

```
message = "Hello %s %s" % (name, surname)
message = f"Hello {name} {surname}"
```

For loops in python

```
monday_temp = [9.1, 8.2, 8.8]
for temperature in monday_temp:
        print (round(temp))

student_grades = {"marry": 9.1, "tom": 9.7}
for grades in student_grades.items():
        print (grades)

for grades in student_grades.keys()
```

prints key

```
for grades in student_grades.values()
```

prints values

Dictionary loops & string formatting

```
1) phone_number = {"john": 1234, "mery": 199
   for pair in phone_number.items():
       message = "{} has number {}".form
                          pair[0], pair[1])
       print(message)
       output: john has number 1234
               mery has number 1999
2) phone_number = {"john": 1234, "merg":
   for key, value in phone number:
       message = "{} has number {}".form
                          (key, value)
       print(message)
```

While loop example

```
1) username = ''
   while username != "pypy"
         username = input("Enter usernam
2) while true:
       username = input("Enter userna
       if username == "pypy":
              break.
       else:
              continue
3) while datetime.datetime.now < date
      datetime.(2021, 10. 8, 19, 30, 10):
   print("its not yet 19:30:20 of 2021:
```

list comprehension

inline for loop

```
temps = [220, 234, 230, 250]
new_temp = [temp/10 for temp in temps]
print(new_temp)
```

with if conditional

```
new_temp = [temp/10 for temp in temps
                if temp != -9999]
```

with if else conditional

```
new_temp = [temp/10 if temp != 9999 else
               0 for temp in temps]
```

syntax is changed.

Convert element in tuple to uppercase for indefinite args

```
def foo(*args):
    args = [x.upper for x in args]
    return sorted(args)
```

function with arbitrary number of keyword args

```
def mean(**kwargs):
    return kwargs
print(mean(a=1, b=2, c=3))
```

non default   default parameter

```
def caboid_volume(a, b, c=10)
    return a*b*c

print(caboid value(2, b=3)
```

non keyword argument   keyword argument

File operation in python

An object is created when we
opening a text file in python

```
myfile = open ("fruits. text")
print (myfile. read())
```

when the file is read the cur
move to the end of the text cont
so when we are printing the file
nothing is shown to overcome this
have to save the file

```
myfile = open(" fruits. txt")
content = myfile . read()
print ( content)
print (content)
```

shows 2 time

closing a file

```
myfile . close()
```

objectname. close

with content manager

```
with open("fruits . txt") as myfile :
        content = myfile . read()
print (content)
```

Different file directory

```
with open ("files / fruits. txt) as my file:
        content = myfile . read()
print ( content)
```

Writing text to a file

```
with open ("files/vegatables.txt", "w") as myfile:
    myfile.write("tomato")
```

if the file already exist python will overwrite the file

'r'	open for reading default
'w'	open for writing overwriting, truncating
'x'	create a new file doesnot overwrite
'a'	open for writing, appending at end.
'b'	binary mode
't'	text mode default
'+'	open a disk for updating (r/w)
'U'	universal newline mode (depreacted)

appending text to an already existing file

```
    with open("files/fruits.txt", "a+")as myfile
        myfile.write("\n orange")
        myfile.seek(0) // put cursor back
        content = myfile.read() at the begining
print(content)
```

when the write method is called orange is appended at last and cursor is positioned last. when print is called without seek(0) cursor is at last and doesnt print anything.

python standard library

Builtin modules   modules written in py

```
>>> import sys      >>> sys.builtin_modul.
(abc, time, import)...
>>> import time
>>> dir(time)
    (sleep, --space-- ... altzone
)
```

There are functions and methods inside python software or interpreter.

eg : Time.sleep(seconds)

~~standard python modules~~

In addition to builtin functions a large num of predefined functions are available as of libraries bundled with python distribution. These functions are in builtin modules. They written in c and integrated with python s

They also contain modules written in python provide standardised solutions some of them h to encourage the portability of python

Eg : os module

```
    >> import os
    >> dir(os)
import time
import os
while true:
    if os.path.exists("files/vegetables.txt"):
        with open("files/vegetables.txt") as f
            print(file.read())
else:
    print("file doesnot exist")
time.sleep()
```

```
f = lamda a,b : a+b
result = f(3,6)
print(result)

nums = [3, 2, 6, 8, 7, 6, 2, 9]
evens = list(filter(lamda n: n%2==0, num
~~odds~~ = ~~list( lamda n:~~
~~odds~~ = list( map( lamda n: n*2, even
double
```

Decorators.

```
    def div(a,b):
            print(a/b)
    def smartdiv(func):
            def inner(~~b~~ a,b):
                    if(a<b)
                        a,b = b,a
            return func(a,b)
        return inner

    div = smart div(div)
    div(2,4)
```

Object Oriented programming

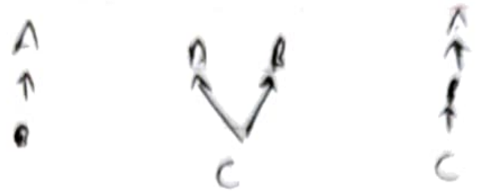
Eg:

```
class computer:
        def config(self):
                print("15gb, 6gb ram")
com1 = computer()
com2 = computer()

com1. config
com2. config
```

Inheritance

The ability to inherit the properties of parent class to child class.

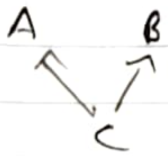
```
class A:
        def feature(self):
                print('1')
class B(A):

        def feature(self)
                print (2)
```



when the contructor is called the python first look for init in child class from which it is called. If not found calls the init methode ~~for~~ of parent class

If the init, is found in child class it is calle called. we can call parent class contructor using super().

In the case of multiple inheritence the <u>methode resolution order</u> is left to right



first call C then A then B.

Polymorphism

ability to take multiple forms

- Duck typing
- methode overloading
- methode overriding
- operator overloading

Duck typing.

Dynamic typing where the type of object is ~~not~~ less important than metho