DATE :__/__/____
PAGE :_____

DATA SCIENCE

PYTHON

Basics

↳ Python is a dynamically typed

C++ is static typed

* print ("Hello", end = " ")
  print ("World")

// Hello World

* print ("Hello", "World", sep = "-")

// Hello-World

* Operators precedence

+, -	left to right
*, /, //	left to right
**	Right to left

* type() → depicts data type of variable

```
a=10
type(a)
    ↳ (class, 'int')
```

DATE : __/__/____
PAGE : _____

Type Conversion

- IMPLICIT
- EXPLICIT

* a = '1001' (base = 2)
  print (int (a, 2)) → convert to decimal from binary

  a = '12'
  print (int (a, 8)) → base-8 (octal → decimal)

  a = 'A1'
  print (int (a, 16)) → base-16 (Hex → decimal)

  ** → exponent operator

Comments
- # (Single line Comments)
- """ ------ """ (Docstring comments)

→ To create an empty set
  S = set ()

DATE : __/__/____
PAGE : _____

Operators

① Arithmatic operators

$x = 9$, $y = 4$

$x + y$	13	Addition
$x - y$	5	Subtraction
$x * y$	36	Multiply
x / y	2.25	Division
$x // y$	2	Floor Division
$x \% 1$	1	Modulo Operator
$x ** y$	6561	Power

* floor (-3.2) → -4
  floor (-5.9) → -6

② Logical Operators

a = 10 b = 20 C = 30

print (a<b and b<C) T AT → True
(a<b) or b>C) T or F → True
(not a>b) not F → True

	AND	OR
T T	T	T
T F	F	T
F T	F	T
F F	F	F

DATE :__/__/__
PAGE :_____

* Expressions that are treated as False:
  None, 0
  Empty string, list, tuple dictionary etc

eg S1 = "" → False
S2 = S1 or "String"
(F or T)

print (S2)
↓
"String"

(F and T) → F
S2 = S1 and "String"
print (S2)
→ "" empty

* x = 10
First is evaluated in or
print (x or 20) // 10
y = 0
print (y or 30) // 30
z = 40
print (z and 50) // 50
↓
last value is evaluated in and

(3) Identity Comparison operators

is , is not

```
x = 10
y = x
print (x is y)      // True
print (x is not y)  // False
```

→ Gives true only for literals like numbers, floats, string

× list, tuple, set ×

```
l1 = [10, 20, 30]
l2 = [10, 20, 30]
print (l1 is l2)
```

↳ False ✓

(4) Membership Test operators

in , not in

string : checks for substring
Dictionary = check for key

list, set, Tuple : check for membership

```
eg S = "geeks"
print ("g" in S)    // True
print ("ee" in S)   // True
print ("gk" in S)   // False
```

DATE : __/__/____
PAGE : _____

d = { 10: "abc", 20: "def" }

print (10 in d) // true (only key)
print ("abc" in d) // false

5

⑤ Bitwise Operators

Bitwise and : &

10

x = 3
y = 6
print (x & y)

// 2

→ 3 : 011
6 : 110 &
2 : 010

15

Bitwise Or : |

x = 3
y = 6
print (x | y)
// 7

011
110 or
111 : 7

20

Bitwise xor : ^

25

Input		output
0	0	0
0	1	1
1	0	1
1	1	0

x = 3
y = 6
print (x ^ y)
// 5

110
011
101
⑤

30

DATE : __/__/__
PAGE :

Left Shift operator <<

```
x=5
print (x<<1)
  //10
(x<<2)   (10100)
  //20
(x<<3)   (101000)
  //40
```

5 : 101
x<<1 → 1010 : 10

Right Shift operator : >>

```
  x=5
print (x>>1)
  //010 (2)
x>>2
  001 (1)
x>>3
  000 (0)
```

5 : 101
x >> 1 → 010

Bitwise not : ~

```
x=5
print(~x)
  //-6
```

Bin of 5 : 00---0101
~5 : 11---1010
2's complement of 6 → 00--110 ← 6
11---1001 ← 1's
01---1010 ← 2's
-6

5000 + (5)2000
10000

DATE : __/__/__
PAGE :

* Arithematic progression nth term in Python

$$n^{th}\ term = a + (n-1)\,d$$

a → first term
d → Common difference

5000, 7000, 90000, 11000 ------

↑ 0

↑ 60

60^{th} term = 5000 + (60)(2000)
= 5000 + 120000
⇒ 125000

```
result = a + (n-1) * d)
print(result)
```

* Geometeric progression

$$n^{th}\ term = a\,r^{n-1}$$

r = common ratio

5000, 10000, 20000 ------ ?

n = 11

11^{th} term = 5000 $(2)^{10}$
= 5000 (1024)
= 5120000

$(1-1)\%7$ $3-1$ $(1+1)\%7$ -10 $5-14$

$n! = (3-2)\%7$ $-9\%7$

DATE : __/__/____
PAGE : _____

res = a**(n-1)

* Program to find first digit of num

```
n = int(input())
while(n>=10)
    n = n/10

print(int(n))
```

Loops in Python

→ Doing something repeatedly.
→ Traversing through collections (list, set, tuple)
→ Running services

① While loop

```
while condition_test:
    statements
    :
    :
```

```
i=0
while(i<5):
    print("Hello!")
    i=i+1
```

→
Hello!
Hello!
Hello!
Hello!
Hello!

DATE : __/__/__
PAGE :

② range() in Python

→
```
r = range(5)
l = list(r)
print(l)
```
↳ [0, 1, 2, 3, 4]

* range(x) : 0 to x-1.

```
type(r)
```
↳ <class, 'range'>

→
```
r = range(10,20)
l = list(r)
print(l)
```
↳ [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

* range(start, stop, step)

③ For loop

```
for x in sequence:
    statements
```

sequence can be list, string, tuple etc

DATE : __/__/____
PAGE : _____

eg l = [10, 20, 30, 40]
for i in l:
    print(i)

// 10
20
30
40

→ S = 'gfg'
for i in S:
    print(i)

// g
f
g

→ for i in range(0, 5, 1)
    print(i)
    ↳

// 0
1
2
3
4

* len(l) → length of list

BREAK	CONTINUE
To exit out of loop	↳ TO continue in loop

NESTED LOOPS

```
for i in range(n)
    for j in range(n)
        print("*", end=" ")
    print()
```

↳ n = 3 →

```
* * *
* * *
* * *
```

To check prime number

```
n = int (input ())
flag = False

for (i) in range (2, n, 1):
        if (n % i == 0):
            flag = True:
            break

if (flag)
    return False
if (n == 1)
    return False
else
    return True
```

To print fibonacci series

```
n = int (input())
f = [0, 1]
for i in range (2, n+1):
        f.append (f[i-1] + f[i-2])
print (f)
```

DATE : __/__/____
PAGE : _____

To capitalize every 1st letter of word

eg hello world → Hello World

```
S = input()
print(S.title())
```

To count no. of words in a sentence using loop

```
count = 1
for i in S:
    if (i == " "):
        count++
print(count)
```

// using split

```
count = len(S.split())
print(count)
```

FUNCTIONS IN PYTHON

→ Block of statements return the specific task

```
def function name(parameter):
    // statement
    // return ex
```

STRINGS

→ sequence of characters
→ store text data
→ Char 'A'-'Z' (ASCII → 65-90)
→ Char 'a'-'z' (ASCII → 97-122)

* print(ord("a"))
  ↳ output 1197

print(chr(97))
  ↳ 'a'

* S = 'geek'
S[0] = g
S[-1] = k

* STRINGS ARE IMMUTABLE
(cannot be changed)

S = 'geek'
S[0] = e
↳ Error

S = """Hello
My name is
Rizwal"""
↓
Multiline

\n → new line

DATE :__/__/____
PAGE :_____

Escape Sequence
↳ \n
To print \n
s = "\\n" // \n

Raw Strings
↳ s1 = r"C:\project\name.py"
↓
// C:\project
name.py ✗

Instead
s1 = r"C:\project\name.py" ✓

To check substring
s1 = "geeks"
s2 = "gee"
print (s2 in s1)
// True

print ("abc" in s1)
// False

DATE : __/__/__
PAGE :

Common operations

① Creating a string

```
S = input ("Enter")
print (S)
```

② Converting str to int

```
S = input("Enter a number")
num = int (S)
print (num)
```

③ To get any letter from str

```
print (S[0], S[3], S[1])
```

④ To calculate length of str

```
print (len(str)
```

OR

```
ln = 0
for i in S:
    ln = ln + 1
print (ln)
```

⑤ To check if a ~~number~~ letter is present / how many times

```
find = input()
count = 0
```

```
for i in s:
    if (i == find)
        count = count + 1

print ("The letter appeared", count, "times")
```

⑥ To replace a letter in String *(s.replace(old,new))

```
rep = input ("Enter the letter u wanna replace")
rep2 = input ("letter to replace with")

inden = 0
str1 = list(str)
for i in str:
    inden = inden + 1
    if (i == rep):
        str1[inden - 1] = rep2

Str = ""
for i in Str1:
    Str += i

print(Str)
```

⑦ To delete a letter

```
// // // Same as above // // //
if (i == num):
    str1.remove (i)

// // // Same // // //
```

```
for i in s:
    if (i == find)
        count = count + 1

print ("The letter appeared", count, "times")
```

⑥ To replace a letter in string *(s.replace(old,new))

```
rep = input ("Enter the letter u wanna replace")
rep2 = input ("letter to replace with")

inden = 0
str1 = list (str)
for i in str:
    inden = inden + 1
    if (i == rep):
        str1[inden - 1] = rep2

str = ""
for i in str1:
    str += i

print(str)
```

⑦ To delete a letter

```
// // // Same as above // // //
if (i == num):
    str1.remove (i)

// // // Same // // //
```

8 Concatenation

```
S1 = "geek"
S2 = "for"
S3 = S1 + S2
print S3
      ↳ //geekfor
```

9) To find index

```
S1 = "geekforgeek"
S2 = "Geek"

print (S1.index(S2))      // 0 (first index)
print (S1.rindex(S2))     // 7 (last index)
print (S1.index(S2, 1, 13))
                  ↓   ↘
                Start  length
```

10) .upper() , .lower()

```
S1 = "Geek"
S1.upper()  → GEEK
S1.lower()  → geek
```

DATE :__/__/____
PAGE :_____

2nd 11) .startswith() , .endswith()

S = "My name"

print (S.startswith("My")) // True
print (S.endswith("name")) // ~~False~~ True

print (S.startswith("Greeks", 1)) // False
index to start

2nd 12) .split() , .join()

Splits into list ~~split~~ make list → string

String to list	list to String
str = 'abcd'	list = [1,2,3,4]
l = str.split()	String = ("".join(list))
or	OR
l = list(str)	String = str.(list)
// [a,b,c,d]	// '1234'

13). .strip() (removes fuancorners)

↓

S = "---Rizwal---"

S.strip("-") // Rizwal

S.lstrip("-") // Rizwal--- (Only left one)

S.rstrip("-") // ---Rizwal (only right)

14) .find()

S1 = "Rizwal"
S2 = "Riz"

print(S1.find(S2))

↳ //0

:(Returns index)

default = -1

NOTE "abc" > "ABC"

↓

(ASCII code)

DATE : __/__/____
PAGE : _____

LIST (mutable)

```
l = [10, 20, 30, 40]

l[0] = 10
l[-1] = 40
```

* To create a list

```
l = []
n = int(input("Enter total numbers"))
for i in range(0,n):
    ele = int(input("Enter"))
    l.append(ele)
print(l)
```

→ max(l), min(l), sum(l)

→ To reverse a list

```
l = [10, 20, 30]
rev = []
for i in range(n-1, -1, -1)
    rev.append(l[i])

print(rev)
```

→ To remove duplicate

```
res = [* set(l)]
print(res)
```

Operations

→ l = [1,2,3]

* l.append(4) // [1,2,3,4]
l.insert(1,5) // [1,5,2,3,4]
print(5 in l) // True
print(l.count(1)) // 1
print(l.index(5)) // 1
l.remove(2) // [1,5,3,4]
l.pop() → // [1,5,3] (last item)
l.pop(0) → // [5,3] (provided index)
del l[1] → // [5]
l.append(6) → // [5,6]
l.reverse() → // [6,5]
l.sort() → // [5,6]

TUPLE (Immutable) (Faster than list)

t = (10,20,"geek")

t[2] = "geek"
t[-1] = "geek"
t[1:2] = (20)
ln(t) = 3
t.count(10) = 1
t.index(20) = 1

DATE : __/__/____
PAGE : _____

SET

→ contains only distinct elements
(no two elements are same)

→ unordered (imp)
→ No ordering
→ Union, intersection, set difference etc are fast.
→ Uses hashing internally

```
S = {10,20,30}
print(S) =      // {10,20,30}        (unordered)
S2 = set([20,30,40])      // {40,20,30}
```

Empty set
S = set()

```
S = {10,20}
* S.add(30)          // {10,20,30}
  S.add(30)          // {10,20,30}
  S.update([40,50])  // {40,10,20,30,50}
  S.discard(30)      // {40,10,20,50}
  S.remove(20)       // {40,10,50}
        ↳ //shows error if ele is not present
  S.clear()          // set()
  del S              //delete all element)
  len(S)             //3
```

DATE : / /
PAGE :

Union operator

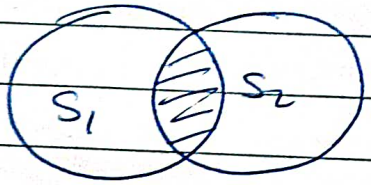
$S_1 = \{1,2,3\}$
$S_2 = \{4,5,6\}$

```
print (S1 | S2)   // {1,2,3,4,5,6}
```

OR

```
S1.union(S2)
```

Intersection



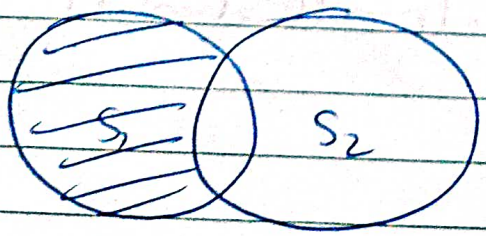
$S_1 = \{1,2,3\}$
$S_2 = \{2,3,4\}$

```
print (S1 & S2)   // {2,3}
```

OR

```
S1.intersection(S2)
```

Difference (S1-S2)




```
print(S1-S2)   // {1}
```

DATE : __/__/__
PAGE : _____

Symmetric difference (^)

print (S1 ^ S2) // {1,4}

* $S_1 = \{2,4,6,8\}$
$S_2 = \{4,8\}$
S_1.~~do~~ isdisjoint (S2) // False
(Because there are common elements in S_1 & S_2)

Subset

$(S_1 <= S_2)$	False	
$(S_1 >= S_2)$	True	(Subset of S_1)
$(S_1 > S_2)$	True	(Proper subset)

DICTIONARY

Key : value

d = {110: "xyz", 101: "abc", 105: "bcd"}

→ collection of key value pair
→ unordered
→ keys must be distinct

Creation

```
d = { }
d['a'] = 1
d['b'] = 2
d['c'] = 3
print(d)          // {'a':1, 'b':2, 'c':3}
print(d['b'])     // 2

print(d.get('a'))   // 1
                    ↳ 2b not present, prints None

OR
print(d.get('d', 'not present'))
                    // not present

d['b'] = 4        // {'a':1, 'b':4, 'c':3}

len(d)   // 3

d.pop('b')    // {'a':1, 'c':3}

d.popitem() → deletes last element
              // {'a':1}
```

Slicing

```
l[start : stop : step]
l = [10, 20, 30, 40, 50]
l[0 : 5 : 2]
    ↳ [10, 30, 50]
```

DATE :__/__/__
PAGE :_____

Comprehensions

* Code to print 10 even numbers

```
l1 = [x for x in range(11) if x%2==0]
print(l1)
    ↳ // [0, 2, 4, 6, 8, 10]
```

* Code to get items of list smaller than x

```
def getsmall(l,x)
    return [e for e in l if e<x]

l = [9,15,12,3,7]
x = 10
print (getsmall(l,x))
    //[9,3,7]
```

// we can make sets with comprehension using {- - - -}

Dictionary comprehension

```
l1 = [1,2,3]
d = {x: x**3 for x in l1}
print(d)                    // {1:1, 2:8, 3:27}
```

* .zip()

```
l1 = [1,2,3]
l2 = [4,5,6]
d = dict(zip(l1,l2)
```

DATE :__/__/__
PAGE :

* Inserting Key : Value → Value : Key

```
d = {1:2, 3:4, 5:6}
d2 = {v:k for (k,v) in d.items()}
print(d2)
```

{2:1, 4:3, 6:5}

OOPS

Object oriented Programming

→ We break code into a set of entities and these entities talk to each other

These entities have data and methods.

Class : blueprint
Object : instance

Class is template for creating objects
→ way to define structure,
→ behaviour of object in program

Self (Keyword) → used to refer current instance of class

DATE : __/__/____
PAGE : _____

* Oan function to write complex number

```
class complex:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def print(self):
        print(str(self.real) + "+i" + str(self.imag))

    def add(self, c):
        self.real += c.real
        self.imag += c.imag
```

object

```
c1 = Complex(10,20)
c1.print()                    # 10+i20

c2 = complex(20,30)
c1.add(c2)
c1.print()                    # 30+i50
```

Encapsulation

→ Bundling of data members and methods.

* We can restrict the variables and methods by making it private or protected.

Protected [Can be accessed through Class & Class function Object]

Single underscore (_)

Private [can be accessed through only Class function]

Double underscore (__).

eg
```
Class A
{
    _a=10
    __b=20
}
```

```
Class B(A)
{
}
```
× Cannot use (a,b) ← as they are private/protected

* In Python every member is accessible everywhere

DATE : __/__/__
PAGE : _____

PRIVATE ACCESS

eg

```
class Test :
    def __init__(self, x, y):
        self.__x = x
        self.y = y

    def __fun(self):
        print("Hi")

t = Test(10,20)
print(t.__x) → // Error
t.__fun() → // Error
```

↓

Private

Decorators

→ Function that takes another function as an argument and enhances behaviour of passed function

```
eg def decfun(f):
       def innerfun():
           print("Welcome")
           f()
       return innerfun

@decfun
def fun():
    print("User")
fun()
```

Class Methods

→ Class methods are used to change class variables internally

```
eg class emp:
       increment = 2
       def __init__(self, salary):
           self.salary = salary

       def increase(self):
           self.salary *= self.increment

       @classmethod
       def change_increment(cls, amount):
           cls.increment = amount
```

DATE : __/__/____
PAGE : _____

```
e = emp (40000)
print (e.salary)          //40000
e.increase()
print (e.salary)          //80000

e.change_increment (3)
e.increase()
print (e.salary)          //240000
```

Static methods

↳ dont have access to class attribute
↳ these are just utility methods

Inheritance

↳ We can call ~~~~ a class inside other class

```
Class program (employee)
        def __init__(self, id, name):
                Super().__init__(id, name)
                self
```

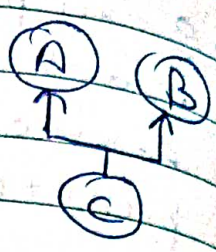
→ used for code reusability
→ Method overriding
→ creates a relation

DATE :__/__/__
PAGE :

Types of Inheritance

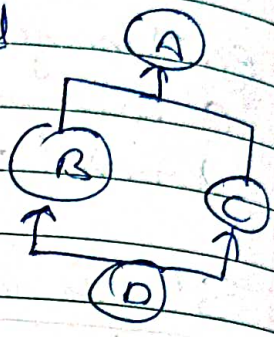
Single

Multiple

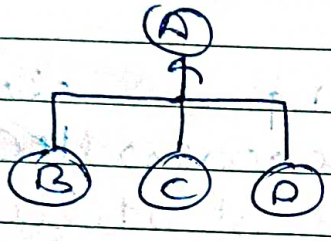


Multilevel

Hybrid



Heirarichal



* Method Overruding

↳ When u have one function in parent class and u create same function with same name in child class

* In Multiple Inheritance

Class stud (name, faculty)

↑ 1st reference is given ↑ then

DATE :__/__/____
PAGE :_____

ABSTRACTION

Abstract class becomes base class and all classes that inherit from it becomes concrete class

Polymorphism

↳ Process that allows same operation to be carried out differently

eg class employee :
    def fun (self):
        print ("Hello 1")

class customer:
    def fun (self):
        print ("Hello 2")

l = [employee() , customer()]
for x in l:
    x.fun()

// Hello 1
Hello 2

→ Method Overriding is part of polymorphism

Operator Overloading

↳ **Dunder methods** → that allows instance of class to interact with built in func and operators
eg (__add__ = 7)

```
class prod:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def __add__(self, other):
        return self.price + other.price

p1 = prod("Key", 600)
p2 = prod("Mouse", 400)
print(p1+p2)
# 1000
```

Dunder methods

- __lt__ → less than
- __le__ → less than equal
- __eq__ → equal
- __ne__ → not equal
- __gt__ → greater than
- __sub__ → subtract
- __mul__ → multiply