Python Tutorial

Syntax and Semantics in Python

- Single line Comments and multiline comments
- Definition of Syntax and Semantics
- Basic Syntax Rules in Python
- Understanding Semantics in Python
- Common Syntax Errors and How to Avoid Them
- Practical Code Examples

Syntax refers to the set of rules that defines the combinations of symbols that are considered to be correctly structured programs in a language. In simpler terms, syntax is about the correct arrangement of words and symbols in a code.

Semantics refers to the meaning or the interpretation of the symbols, characters, and commands in a language. It is about what the code is supposed to do when it runs.

In [ ]:
```
## Basic Syntax Rules In Python
## Case sensitivity- Python is case sensitive

name="Sanad"
Name="Singh"

print(name)
print(Name)
```

```
Sanad
Singh
```

Indentation

Indentation in Python is used to define the structure and hierarchy of the code.

Unlike many other programming languages that use braces {} to delimit blocks of code, Python uses indentation to determine the grouping of statements. This means that all the statements within a block must be indented at the same level.

In [ ]:
```
## Indentation
## Python uses indentation to define blocks of code. Consistent use of spaces (comm


age=32
if age>30:
  print(age)
print(age)
```

```
32
32
```

In [ ]:
```
## This is a single line comment
print("Hello World")
```

```
Hello World
```

In [ ]:
```
## Line Continuation

##Use a backslash (\) to continue a statement to the next line

total=1+2+3+4+5+6+7+\
4+5+6

print(total)
```

```
43
```

In [ ]:
```
## Multiple Statements on a single line
x=5;y=10;z=x+y
print(z)
```

```
15
```

In [ ]:
```
##Understand  Semnatics In Python
# variable assignment

age=25 ## integer
name="Sanad" ## string
```

In [ ]:
```
type(age)
```

Out[ ]:
```
int
```

In [ ]:
```
type(name)
```

Out[ ]:
```
str
```

In [ ]:
```
## Type Inference
var=10
print(type(var))
var="Sanad"
print(type(var))
```

```
<class 'int'>
<class 'str'>
```

In [ ]:
```
age=32
if age>30:
    print(age)
```

```
32
```

Errors

In [ ]:
```
## Name Error
a=b
```

In [ ]:
```
## Code exmaples of indentation
if True:
    print("Correct Indentation")
    if False:
        print("This will not print")
    print("This will print")
print("Outside the if block")
```

```
Correct Indentation
This will print
Outside the if block
```

Key Points:

Understanding the syntax and semantics of Python is crucial for writing correct and meaningful programs.

Syntax ensures the code is properly structured, while semantics ensures the code behaves as expected. Mastering these concepts will help in writing efficient and error-free Python code.

Variables

Variables are fundamental elements in programming used to store data that can be referenced and manipulated in a program.

In Python, variables are created when you assign a value to them, and they do not need explicit declaration to reserve memory space.

The declaration happens automatically when you assign a value to a variable.

- Declaring and Assigning Variables
- Naming Conventions
- Understanding Variable Types
- Type Checking and Conversion
- Dynamic Typing
- Examples and Common Errors

In [ ]:
```
## Declaring And Assigning Variables

name="Sanad"
age=25
height=5.5
student=False

## printing the variables

print("Name:",name)
print("age :",age)
print("Height:",height)
```

```
Name: Sanad
age : 25
Height: 5.5
```

In [ ]:
```
## Naming Conventions
## Variable names should be descriptive
## They must start with a letter or an '_' and contains letter,numbers and underscc
## variables names case sensitive

#valid variable names

first_name="Sanad"
last_name="Singh"
```

In [ ]:
```
print(first_name,"",last_name)
```

```
Sanad  Singh
```

Invalid variable names

2age=30

first-name="sanad"

@name="singh"

In [ ]:
```
2age=30
```

```
  File "<ipython-input-108-b94744f93b18>", line 1
    2age=30
    ^
SyntaxError: invalid decimal literal
```

In [ ]:
```
first-name="sanad"
```

```
  File "<ipython-input-109-6c7a5ab5d29b>", line 1
    first-name="sanad"
    ^
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of
'='?
```

In [ ]:
```
@name="singh"
```

```
  File "<ipython-input-110-6ed4f45cfbc3>", line 1
    @name="singh"
     ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

In [ ]:
```
## Understnading Variable types
## Python is dynamically typed,type of a variable is determined at runtime

name="Sanad" #str
age=25 #int
height=5.5 #float
student=False #bool

print(type(name))
type(height)
```

Out[ ]:
```
<class 'str'>
float
```

Type conversion

In [ ]:
```
age=25
print(type(age))
```

```
<class 'int'>
```

In [ ]:
```
age_str=str(age)

print(age_str)
```

```
print(type(age_str))
```

```
25
<class 'str'>
```

In [ ]:
```
age='25'
print(type(int(age)))
```

```
<class 'int'>
```

In [ ]:
```
name="Sam"
int(name)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-115-5a332d30caa1> in <cell line: 2>()
      1 name="Sam"
----> 2 int(name)

ValueError: invalid literal for int() with base 10: 'Sam'
```

In [ ]:
```
height=5.11
type(height)
```

Out[ ]:
```
float
```

In [ ]:
```
float(int(height))
```

Out[ ]:
```
5.0
```

In [ ]:
```
## Dynamic Typing
## During program execution, python allows to change the type of a variable

var=102 #int
print(var,type(var))

var="Rambo"
print(var,type(var))

var=3.14
print(var,type(var))
```

```
102 <class 'int'>
Rambo <class 'str'>
3.14 <class 'float'>
```

In [ ]:
```
## input

age=int(input("What is the age"))
print(age,type(age))
```

```
What is the age20
20 <class 'int'>
```

Simple Calculator using basic Arithmetic operations such as

- addition
- subtraction
- multiplication

- division

In [ ]:
```
### Simple calculator
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

sum = num1 + num2
difference = num1 - num2
product = num1 * num2
quotient = num1 / num2

print("Sum:", sum)
print("Difference:", difference)
print("Product:", product)
print("Quotient:", quotient)
```

```
Enter first number: 50
Enter second number: 50
Sum: 100.0
Difference: 0.0
Product: 2500.0
Quotient: 1.0
```

KKB (Kaam ki Baat):

- Variables are essential in Python programming for storing and manipulating data.
- Understanding how to declare, assign, and use variables effectively is crucial for writing functional and efficient code.
- Following proper naming conventions and understanding variable types will help in maintaining readability and consistency in your code.

DataTypes

1. **Definition**:

- Data types are a classification of data which tell the compiler or interpreter how the programmer intends to use the data.
- They determine the type of operations that can be performed on the data, the values that the data can take, and the amount of memory needed to store the data.

2. ***Importance of Data Types in Programming***

Explanation:

- Data types ensure that data is stored in an efficient way.
- They help in performing correct operations on data.
- Proper use of data types can prevent errors and bugs in the program.

Basic Data Types

- Integers
- Floating-point numbers
- Strings
- Booleans

Advanced Data Types

- Lists
- Tuples
- Sets
- Dictionaries

Type Conversion

In [ ]:
```
## Integer Example
age=28
type(age)
```

Out[ ]:
```
int
```

In [ ]:
```
##floating point datatype
height=5.6
print(height)
print(type(height))
```

```
5.6
<class 'float'>
```

In [ ]:
```
## string datatype example
name="Sanad"
print(name)
print(type(name))
```

```
Sanad
<class 'str'>
```

In [ ]:
```
## boolean datatype
is_true=True
type(is_true)
```

Out[ ]:
```
bool
```

In [ ]:
```
a=10
b=10
type(a==b)
```

Out[ ]:
```
bool
```

In [ ]:
```
## common errors

result="Hello" + 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-126-3a0769e33931> in <cell line: 3>()
      1 ## common errors
      2
----> 3 result="Hello" + 5

TypeError: can only concatenate str (not "int") to str
```

In [ ]:
```
result="Hello" + str(5)
print(result)
```

```
Hello5
```

Deep Dive into Operators

Arithmetic Operators

- Addition
- Subtraction
- Multiplication
- Division
- Floor Division
- Modulus
- Exponentiation

Comparison Operators

- Equal to
- Not equal to
- Greater than
- Less than
- Greater than or equal to
- Less than or equal to

Logical Operators

- AND
- OR
- NOT

Practical Examples and Common Errors

In [ ]:
```
## Arithmethic Operation

a= 200
b = 12

add=a+b  #addiiton
sub=a-b  #substraction
mult=a*b #multiplication
div=a/b  #division
floor_div=a//b ## floor division
modulus=a%b #modulus operation

exponent=a**b ## Exponentiation

print(add)
print(sub)
print(mult)
print(div)
print(floor_div)
print(modulus)
print(exponent)
```

```
212
188
2400
16.666666666666668
16
8
4096000000000000000000000000
```

In [ ]:
```
print(1035/5)
print(210/5)
print(216//5)
```

```
207.0
42.0
43
```

Comparison Operators

In [ ]:
```
## Comparison Operators
## == Equal to
a=10
b=10

a==b
```

Out[ ]:
```
True
```

In [ ]:
```
str1="Pankaj"
str2="Raj"

str1==str2
```

Out[ ]:
```
False
```

In [ ]:
```
## Not Equal to !=
str1!=str2
```

Out[ ]:
```
True
```

In [ ]:
```
str3="Sanad"
str4="Sanad"

str3!=str4
```

Out[ ]:
```
False
```

In [ ]:
```
# greater than >

num1=45
num2=55

num1>num2
```

Out[ ]:
```
False
```

In [ ]:
```
## less than <

print(num1<num2)
```

```
True
```

In [ ]:
```
#greater than or equal to
number1=45
number2=45

print(number1>=number2)
```

```
True
```

In [ ]:
```
#less than or equal to
number1=44
number2=45

print(number1<=number2)
```

```
True
```

Logical Operators

In [ ]:
```
## And ,Not,OR
X=True
Y=True

result =X and Y
print(result)
```

```
True
```

In [ ]:
```
X=False
Y=True

result =X and Y
print(result)
```

```
False
```

In [ ]:
```
## OR
X=False
Y=False

result =X or Y
print(result)
```

```
False
```

In [ ]:
```
# Not operator
X=False
not X
```

Out[ ]:
```
True
```

In [ ]:
```
# Simple calculator using input function

num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

# Performing arithmetic operations

addition = num1 + num2
subtraction = num1 - num2
multiplication = num1 * num2
division = num1 / num2
floor_division = num1 // num2
modulus = num1 % num2
exponentiation = num1 ** num2
```

```
# Displaying results

print("Addition:", addition)
print("Subtraction:", subtraction)
print("Multiplication:", multiplication)
print("Division:", division)
print("Floor Division:", floor_division)
print("Modulus:", modulus)
print("Exponentiation:", exponentiation)
```

```
Enter first number: 35
Enter second number: 10
Addition: 45.0
Subtraction: 25.0
Multiplication: 350.0
Division: 3.5
Floor Division: 3.0
Modulus: 5.0
Exponentiation: 2758547353515625.0
```

Python Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello"

In [ ]:
```
print("Hello")
print('Hello')
```

```
Hello
Hello
```

In [ ]:
```
a = "Hello"   #Assign String to a Variable
print(a)
```

```
Hello
```

In [ ]:
```
b="world"
print(b)
```

```
world
```

In [ ]:
```
c=a+b  #concatination
```

In [ ]:
```
print(c)
```

```
Helloworld
```

In [ ]:
```
#Multiline Strings

a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
- However, Python does not have a character data type, a single character is simply a string with a length of 1.

In [ ]:
```
# Get the character at position 1 (remember that the first character has the positi

a = "Hello, World!"
print(a[1])
```

```
e
```

In [ ]:
```
# get the length of string

s="spam"
len(s)
```

Out[ ]:
```
4
```

In [ ]:
```
#pull out p, m using indexing

print(s[1])

print(s[-1])
```

```
p
m
```

In [ ]:
```
s[len(s)-1] #reverse Indexing
```

Out[ ]:
```
'm'
```

In [ ]:
```
# pull out pa, pam,spa,spam using slicing

print(s[1:3])
print(s[1:4])
```

```
pa
pam
```

In [ ]:
```
print(s[:-1]) # eliminate the last element of the string
print(s[:]) #all character from string
```

```
spa
spam
```

Concatination (adding of two strings)

In [ ]:
```
s
```

Out[ ]:
```
'spam'
```

In [ ]:
```
s+ "email"
```

Out[ ]:
```
'spamemail'
```

In [ ]:
```
print(s + "email")
```

```
spamemail
```

In [ ]:
```
s**2   #power operation doen't work in string
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-158-a03723da0053> in <cell line: 1>()
----> 1 s**2   #power operation doen't work in string

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

In [ ]:
```
s[0]='z'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-159-ab471980a0a4> in <cell line: 1>()
----> 1 s[0]='z'

TypeError: 'str' object does not support item assignment
```

In [ ]:
```
S="shruberry"
```

In [ ]:
```
S[1:] + "c" #remove s from S and add c within the string
```

Out[ ]:
```
'hruberryc'
```

In [ ]:
```
s[:1]  #pull out s from the string
```

Out[ ]:
```
's'
```

In [ ]:
```
s[1:2] + "c"  #pull p and add c within the string
```

Out[ ]:
```
'pc'
```

In [ ]:
```
S[1:2] +"c"
```

Out[ ]:
```
'hc'
```

In [ ]:
```
s[0]+"c"+S[2:9] #addition of two string including one seprate string
```

Out[ ]:
```
'scruberry'
```

In [ ]:
```
S="shruberry"  #convert sting to list
l= list(S)
print (l)
```

```
['s', 'h', 'r', 'u', 'b', 'e', 'r', 'r', 'y']
```

In [ ]:
```
#replace h with c within the list S
l[1]="c"
print(l)
```

```
['s', 'c', 'r', 'u', 'b', 'e', 'r', 'r', 'y']
```

In [ ]:
```
# again convert the list to the string using join

''.join(l)
```

Out[ ]:
```
'scruberry'
```

In [ ]:
```
B=bytearray(b"hima")
B.extend(b"nchal")
#print(B)
B.decode()
```

Out[ ]:
```
'himanchal'
```

In [ ]:
```
chr("A")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-170-55562c7d81ed> in <cell line: 1>()
----> 1 chr("A")

TypeError: 'str' object cannot be interpreted as an integer
```

In [ ]:
```
chr("698")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-171-ea7c9a6e0dcd> in <cell line: 1>()
----> 1 chr("698")

TypeError: 'str' object cannot be interpreted as an integer
```

In [ ]:
```
chr(65)
```

Out[ ]:
```
'A'
```

In [ ]:
```
ord ('A')
```

Out[ ]:
```
65
```

In [ ]:
```
chr (32)
```

Out[ ]:
```
' '
```

Implement the Loop within the string

In [ ]:
```
myjob="hacker"
for item in myjob:    #loop syntax how to apply loop answer will be in vertical
  print(item)
```

```
h
a
c
k
e
r
```

In [ ]:
```
abc123="hello"  # valid identifier
```

In [ ]:
```
123abc="hello"# invalid identifier
```

```
  File "<ipython-input-177-c1f7b34dc63e>", line 1
    123abc="hello"# invalid identifier
      ^
SyntaxError: invalid decimal literal
```

In [ ]:
```
123 abc="hello"  # invalid identifier
```

```
  File "<ipython-input-178-e58bc21c7d72>", line 1
    123 abc="hello"  # invalid identifier
        ^
SyntaxError: invalid syntax
```

In [ ]:
```
myjob="hacker"
for item in myjob:    #loop syntax how to apply loop answer will be in horizontal
  print (item,end=" ")   #attribute on print function end=
```

```
h a c k e r
```

In [ ]:
```
s="spam" #answer will be in tuple ()

s[0],s[1]
```

Out[ ]:
```
('s', 'p')
```

In [ ]:
```
s="spam"
s[1:3],s[1:4],s[0:3]
```

Out[ ]:
```
('pa', 'pam', 'spa')
```

In [ ]:
```
a=('s', 'p')
type(a)
```

Out[ ]:
```
tuple
```

In [ ]:
```
s="abcdefghijklmnop"
s[1]+s[3]+s[5]+s[7]+s[9]
```

Out[ ]:
```
'bdfhj'
```

In [ ]:
```
s="abcdefghijklmnop" #stepping range is defined
s[1:10:2]
```

Out[ ]:
```
'bdfhj'
```

In [ ]:
```
s="abcdefghijklmnop"  # syntaz [start:end:step]
s[0::2]
```

Out[ ]:
```
'acegikmo'
```

In [ ]:
```
str1="Indian"   #reverse indexing
str1[::-1]
```

Out[ ]:
```
'naidnI'
```

In [ ]:
```
str2="abcdefg"  #reverse indexing along with the stepping
str2[-2:-6:-1]
```

Out[ ]:
```
'fedc'
```

In [ ]:
```
int("42"), str(42)
```

Out[ ]:
```
(42, '42')
```

In [ ]:
```
float("42"), str(42)
```

Out[ ]:
```
(42.0, '42')
```

In [ ]:
```
s="spam"
s+"email!"
```

Out[ ]:
```
'spamemail!'
```

In [ ]:
```
s="spamemail!"        #replace syntax  replacing old substring with new sub string
s.replace ("email","hamburger")
```

Out[ ]:
```
'spamhamburger!'
```

In [ ]:
```
print("that is %d %s bird!" % (100, "dead")) #formatting
```

```
that is 100 dead bird!
```

In [ ]:
```
print ("that is {0} {1} {2}!".format(100, "dead","bird"))
```

```
that is 100 dead bird!
```

Conditional Statements (if, elif, else)

- if Statement
- else Statement
- elif Statement
- Nested Conditional Statements
- Practical Examples
- Common Errors and Best Practices

In [ ]:
```
## if statement
age=20

if age>=18:
    print("You are allowed to vote in the elections")

age>=18
```

Out[ ]:
```
You are allowed to vote in the elections
True
```

In [ ]:
```
## if statement
age=float(input("Enter the age"))

if age>=18:
    print("You are allowed to vote in the elections")
```

```
Enter the age18.2
You are allowed to vote in the elections
```

ELSE

The else statement executes a block of code if the condition in the if statement is False.

In [ ]:
```
age=16

if age>=18:
    print("You are eligible for voting")
else:
    print("You are a minor")
```

```
You are a minor
```

In [ ]:
```
age=float(input("Enter the age"))

if age>=18:
    print("You are eligible for voting")
else:
    print("You are a minor")
```

```
Enter the age20
You are eligible for voting
```

In [ ]:
```
age=float(input("Enter the age"))

if age>=18:
    print("You are eligible for voting")
else:
    print("You are a minor")
```

```
Enter the age17.9
You are a minor
```

elif

The elif statement allows you to check multiple conditions. It stands for "else if"

In [ ]:
```
age=17

if age<13:
    print("You are a child")
elif age<18:
    print("You are a teenager")
else:
    print("You are an adult")
```

```
You are a teenager
```

In [ ]:
```
## Nested Condiitonal Statements

# You can place one or more if, elif, or else statements inside another if, elif, c

## number even ,odd,negative

num=int(input("Enter the number"))

if num>0:
    print("The number is positive")
    if num%2==0:
        print("The number is even")
    else:
        print("The number is odd")

else:
    print("The number is zero or negative")
```

```
Enter the number20
The number is positive
The number is even
```

In [ ]:
```
## Nested Condiitonal Statements

# You can place one or more if, elif, or else statements inside another if, elif, c

## number even ,odd,negative

num=int(input("Enter the number"))

if num>0:
    print("The number is positive")
    if num%2==0:
        print("The number is even")
    else:
        print("The number is odd")

else:
    print("The number is zero or negative")
```

```
Enter the number41
The number is positive
The number is odd
```

In [ ]:
```
## Practical Examples

## Determine if a year is a leap year using nested condition statement

year=int(input("Enter the year"))

if year%4==0:
    if year%100==0:
        if year%400==0:
            print(year,"is a leap year")
        else:
            print(year,"is not a leap year")
    else:
        print(year,"is a leap year")

else:
    print(year,"is not a leap year")
```

```
Enter the year200
200 is not a leap year
```

In [ ]:
```
## Practical Examples

## Determine if a year is a leap year using nested condition statement

year=int(input("Enter the year"))

if year%4==0:
    if year%100==0:
        if year%400==0:
            print(year,"is a leap year")
        else:
            print(year,"is not a leap year")
    else:
        print(year,"is a leap year")

else:
    print(year,"is not a leap year")
```

```
Enter the year204
204 is a leap year
```

In [ ]:
```python
# Simple Calculator program using the conditional statement
# Take user input

num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
operation = input("Enter operation (+, -, *, /): ")

# Perform the requested operation
if operation == '+':
    result = num1 + num2
elif operation == '-':
    result = num1 - num2
elif operation == '*':
    result = num1 * num2
elif operation == '/':
    if num2 != 0:
        result = num1 / num2
    else:
        result = "Error! Division by zero."
else:
    result = "Invalid operation."

print("Result:", result)
```

```
Enter first number: 10
Enter second number: 14
Enter operation (+, -, *, /): /
Result: 0.7142857142857143
```

In [ ]:
```python
### Determine the ticket price based on age and whether the person is a student.

# Ticket pricing based on age and student status

# Take user input
age = int(input("Enter your age: "))
is_student = input("Are you a student? (yes/no): ").lower()

# Determine ticket price
if age < 5:
    price = "Free"
elif age <= 12:
    price = "10"
elif age <= 17:
    if is_student == 'yes':
        price = "12"
    else:
        price = "15"
elif age <= 64:
    if is_student == 'yes':
        price = "$18"
    else:
        price = "25"
else:
    price = "20"

print("Ticket Price:", price)
```

```
Enter your age: 20
Are you a student? (yes/no): yes
Ticket Price: $18
```

In [ ]:
```
age = int(input("Enter your age: "))
is_student = input("Are you a student? (yes/no): ").lower()

# Determine ticket price
if age < 5:
    price = "Free"
elif age <= 12:
    price = "10"
elif age <= 17:
    if is_student == 'yes':
        price = "12"
    else:
        price = "15"
elif age <= 64:
    if is_student == 'yes':
        price = "$18"
    else:
        price = "25"
else:
    price = "20"

print("Ticket Price:", price)
```

```
Enter your age: 10
Are you a student? (yes/no): no
Ticket Price: 10
```

Employee Bonus Calculation

Calculate an employee's bonus based on their performance rating and years of service.

In [ ]:
```
# Take user input
service = int(input("Enter years of service: "))
performance= float(input("Enter performance rating (1.0 to 5.0): "))

# Determine bonus percentage
if performance >= 4.5:
    if service > 10:
        bonus_percentage = 20
    elif service > 5:
        bonus_percentage = 15
    else:
        bonus_percentage = 10
elif performance >= 3.5:
    if service > 10:
        bonus_percentage = 15
    elif service > 5:
        bonus_percentage = 10
    else:
        bonus_percentage = 5
else:
    bonus_percentage = 0

# Calculate bonus amount
salary = float(input("Enter current salary: "))
bonus_amount = salary * bonus_percentage / 100

print("Bonus Amount: ${:.2f}".format(bonus_amount))
```

```
Enter years of service: 5
Enter performance rating (1.0 to 5.0): 4
Enter current salary: 2000
Bonus Amount: $100.00
```

In [ ]:
```
service = int(input("Enter years of service: "))
performance= float(input("Enter performance rating (1.0 to 5.0): "))

# Determine bonus percentage
if performance >= 4.5:
    if service > 10:
        bonus_percentage = 20
    elif service > 5:
        bonus_percentage = 15
    else:
        bonus_percentage = 10
elif performance >= 3.5:
    if service > 10:
        bonus_percentage = 15
    elif service > 5:
        bonus_percentage = 10
    else:
        bonus_percentage = 5
else:
    bonus_percentage = 0

# Calculate bonus amount
salary = float(input("Enter current salary: "))
bonus_amount = salary * bonus_percentage / 100

print("Bonus Amount: ${:.2f}".format(bonus_amount))
```

```
Enter years of service: 6
Enter performance rating (1.0 to 5.0): 5
Enter current salary: 1000
Bonus Amount: $150.00
```

User Login System

A simple user login system that checks the username and password.

In [ ]:
```
# User login system

# Predefined username and password
stored_username = "admin"
stored_password = "password123"

# Take user input
username = input("Enter username: ")
password = input("Enter password: ")

# Check login credentials
if username == stored_username:
    if password == stored_password:
        print("Login successful!")
    else:
        print("Incorrect password.")
else:
    print("Username not found.")
```

```
Enter username: sanad
Enter password: singh
Username not found.
```

In [ ]:
```
stored_username = "admin"
stored_password = "password123"
```

```
# Take user input
username = input("Enter username: ")
password = input("Enter password: ")

# Check login credentials
if username == stored_username:
    if password == stored_password:
        print("Login successful!")
    else:
        print("Incorrect password.")
else:
    print("Username not found.")
```

```
Enter username: admin
Enter password: password123
Login successful!
```

Loops

for Loop

- Iterating over a range
- Iterating over a string

while Loop

Loop Control Statements

```
- break
- continue
- pass
```

Nested Loops

In [ ]:
```
range(5)

## for loop

for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [ ]:
```
for i in range(1,6):
    print(i)
```

```
1
2
3
4
5
```

In [ ]:
```
for i in range(1,10,2): #Start:End:Step
    print(i)
```

```
1
3
5
7
9
```

In [ ]:
```
for i in range(10,1,-1): #reverse stepping
    print(i)
```

```
10
9
8
7
6
5
4
3
2
```

In [ ]:
```
for i in range(10,1,-2): #reverse stepping
    print(i)
```

```
10
8
6
4
2
```

In [ ]:
```
## strings

str="Sanad Kumar Singh"

for i in str:
    print(i)
```

```
S
a
n
a
d

K
u
m
a
r

S
i
n
g
h
```

In [ ]:
```
## while loop

## The while loop continues to execute as long as the condition is True.

count=0

while count<5:
    print(count)
    count=count+1
```

```
0
1
2
3
4
```

Loop Control Statements

In [ ]:
```python
## break
## The break statement exits the loop permaturely

## break sstatement

for i in range(10):
    if i==5:
        break
    print(i)
```

```
0
1
2
3
4
```

Continue

In [ ]:
```python
## The continue statement skips the current iteration and continues with the next.

for i in range(10):
    if i%2==0:
        continue
    print(i)
```

```
1
3
5
7
9
```

Pass

In [ ]:
```python
## The pass statement is a null operation; it does nothing.

for i in range(5):
    if i==3:
        pass
    print(i)
```

```
0
1
2
3
4
```

In [ ]:
```python
## Nested loopss
## a loop inside a loop

for i in range(3):
    for j in range(2):
        print(f"i:{i} and j:{j}")
```

```
i:0 and j:0
i:0 and j:1
i:1 and j:0
i:1 and j:1
i:2 and j:0
i:2 and j:1
```

Calculate the sum of first N natural numbers using a while and for loop

In [ ]:
```
## while loop

n=10
sum=0
count=1

while count<=n:
    sum=sum+count
    count=count+1

print("Sum of first 10 natural number:",sum)
```

```
Sum of first 10 natural number: 55
```

In [ ]:
```
n=10
sum=0
for i in range(11):
    sum=sum+i

print(sum)
```

```
55
```

Prime numbers between 1 and 100

In [ ]:
```
for num in range(1,101):
    if num>1:
        for i in range(2,num):
            if num%i==0:
                break
        else:
            print(num)
```

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```

KKB :

Loops are powerful constructs in Python that allow you to execute a block of code multiple times.

By understanding and using for and while loops, along with loop control statements like break, continue, and pass, you can handle a wide range of programming tasks efficiently.

Introduction To Lists

- Lists are ordered, mutable collections of items.
- They can contain items of different data types.

1. Introduction to Lists
2. Creating Lists
3. Accessing List Elements
4. Modifying List Elements
5. List Methods
6. Slicing Lists
7. Iterating Over Lists
8. List Comprehensions
9. Nested Lists
10. Practical Examples and Common Errors

In [ ]:
```
lst=[]
print(type(lst))
```

```
<class 'list'>
```

In [ ]:
```
names=["Sanad","Singh","RAJ",1,2,3,4,5]
print(names)
```

```
['Sanad', 'Singh', 'RAJ', 1, 2, 3, 4, 5]
```

In [ ]:
```
mixed_list=[1,"Hello",3.14,True,"World"]
print(mixed_list)
```

```
[1, 'Hello', 3.14, True, 'World']
```

In [ ]:
```
### Accessing List Elements Using Indexing, reverse indexing And Slicing

fruits=["apple","banana","cherry","kiwi","gauva"]
```

In [ ]:
```
print(fruits[0])
print(fruits[2])
print(fruits[4])
print(fruits[-1])
```

```
apple
cherry
gauva
gauva
```

In [ ]:
```
print(fruits[1:])
print(fruits[1:3])
```

```
['banana', 'cherry', 'kiwi', 'gauva']
['banana', 'cherry']
```

In [ ]:
```
## Modifying The List elements
fruits
fruits[1]="watermelon"
print(fruits)
```

```
['apple', 'watermelon', 'cherry', 'kiwi', 'gauva']
```

In [ ]:
```
fruits[1:]="watermelon"
fruits
```

Out[ ]:
```
['apple', 'w', 'a', 't', 'e', 'r', 'm', 'e', 'l', 'o', 'n']
```

In [ ]:
```
fruits=["apple","banana","cherry","kiwi","gauva"]
```

In [ ]:
```
## List Methods

fruits.append("orange") ## Add an item to the end
print(fruits)
```

```
['apple', 'banana', 'cherry', 'kiwi', 'gauva', 'orange']
```

In [ ]:
```
fruits.insert(1,"watermelon") ## Inserting new data at index no. 1
print(fruits)
```

```
['apple', 'watermelon', 'banana', 'cherry', 'kiwi', 'gauva', 'orange']
```

In [ ]:
```
fruits.remove("banana") ## Removing the first occurance of an item
print(fruits)
```

```
['apple', 'watermelon', 'cherry', 'kiwi', 'gauva', 'orange']
```

In [ ]:
```
## Remove and return the last element
popped_fruits=fruits.pop()
print(popped_fruits)
print(fruits)
```

```
orange
['apple', 'watermelon', 'cherry', 'kiwi', 'gauva']
```

In [ ]:
```
index=fruits.index("cherry")
print(index)
```

```
2
```

In [ ]:
```
fruits.insert(2,"banana")
print(fruits.count("banana"))
```

```
1
```

In [ ]:
```
fruits
```

Out[ ]:
```
['apple', 'watermelon', 'banana', 'cherry', 'kiwi', 'gauva']
```

In [ ]:
```
fruits.sort() ## SSorts the list in ascending order
```

In [ ]:
```
fruits
```

Out[ ]:
```
['apple', 'banana', 'cherry', 'gauva', 'kiwi', 'watermelon']
```

In [ ]:
```
fruits.reverse() ## Reverse the list
```

In [ ]:
```
fruits
```

Out[ ]:
```
['watermelon', 'kiwi', 'gauva', 'cherry', 'banana', 'apple']
```

In [ ]:
```
fruits.clear() ## Remove all items from the list

print(fruits)
```

```
[]
```

In [ ]:
```
## Slicing List
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(numbers[2:5])
print(numbers[:5])
print(numbers[5:])
print(numbers[::2])
print(numbers[::-1])
```

```
[3, 4, 5]
[1, 2, 3, 4, 5]
[6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [ ]:
```
numbers[::3]
```

Out[ ]:
```
[1, 4, 7, 10]
```

In [ ]:
```
numbers[::-2]
```

Out[ ]:
```
[10, 8, 6, 4, 2]
```

In [ ]:
```
### Iterating Over List using for loop

for number in numbers:
    print(number)
```

```
1
2
3
4
5
6
7
8
9
10
```

In [ ]:
```python
## Iterating with index
for index,number in enumerate(numbers):
    print(index,number)
```

```
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9 10
```

In [ ]:
```python
## List comprehension
lst=[]
for x in range(10):
    lst.append(x**2)

print(lst)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [ ]:
```python
[x**2 for x in range(10)]
```

Out[ ]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension

- Basics Syantax---------[expression for item in iterable]
- with conditional logic---------[expression for item in iterable if condition]
- Nested List Comprehension--------[expression for item1 in iterable1 for item2 in iterable2]

In [ ]:
```python
### Basic List Comphrension

sqaure=[num**2 for num in range(10)]
print(sqaure)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [ ]:
```python
### List Comprehension with Condition
lst=[]
for i in range(10):
    if i%2==0:
        lst.append(i)

print(lst)
```

```
[0, 2, 4, 6, 8]
```

In [ ]:
```
even_numbers=[num for num in range(10) if num%2==0]
print(even_numbers)
```

```
[0, 2, 4, 6, 8]
```

In [ ]:
```
## Nested List Comphrension

lst1=[1,2,3,4]
lst2=['a','b','c','d']

pair=[[i,j] for i in lst1 for j in lst2]

print(pair)
```

```
[[1, 'a'], [1, 'b'], [1, 'c'], [1, 'd'], [2, 'a'], [2, 'b'], [2, 'c'], [2, 'd'],
[3, 'a'], [3, 'b'], [3, 'c'], [3, 'd'], [4, 'a'], [4, 'b'], [4, 'c'], [4, 'd']]
```

In [ ]:
```
## List Comprehension with function calls
words = ["hello", "world", "python", "list", "comprehension"]
lengths = [len(word) for word in words]
print(lengths)  # Output: [5, 5, 6, 4, 13]
```

```
[5, 5, 6, 4, 13]
```

KKB

- List comprehensions are a powerful and concise way to create lists in Python.
- They are syntactically compact and can replace more verbose looping constructs.
- Understanding the syntax of list comprehensions will help you write cleaner and more efficient Python code.

Introduction to Tuples

Explanation:

- Tuples are ordered collections of items that are immutable.
- They are similar to lists, but their immutability makes them different.

Things to learn

- Creating Tuples
- Accessing Tuple Elements
- Tuple Operations
- Immutable Nature of Tuples
- Tuple Methods
- Packing and Unpacking Tuples
- Nested Tuples

In [ ]:
```
## creating a tuple
empty_tuple=()
print(empty_tuple)
print(type(empty_tuple))
```

```
()
<class 'tuple'>
```

In [ ]:
```
lst=list()
print(type(lst))
tpl=tuple()
print(type(tpl))
```

```
<class 'list'>
<class 'tuple'>
```

In [ ]:
```
numbers=tuple([1,2,3,4,5,6])
numbers
```

Out[ ]:
```
(1, 2, 3, 4, 5, 6)
```

In [ ]:
```
list((1,2,3,4,5,6))
```

Out[ ]:
```
[1, 2, 3, 4, 5, 6]
```

In [ ]:
```
mixed_tuple=(1,"Hello World",3.14, True)
print(mixed_tuple)
```

```
(1, 'Hello World', 3.14, True)
```

Accessing Tuple Elements

In [ ]:
```
numbers
```

Out[ ]:
```
(1, 2, 3, 4, 5, 6)
```

In [ ]:
```
print(numbers[2])
print(numbers[-1])
```

```
3
6
```

In [ ]:
```
numbers[0:4]
```

Out[ ]:
```
(1, 2, 3, 4)
```

In [ ]:
```
numbers[::-1]
```

Out[ ]:
```
(6, 5, 4, 3, 2, 1)
```

In [ ]:
```
## Tuple Operations

concatenation_tuple=numbers + mixed_tuple
print(concatenation_tuple)
```

```
(1, 2, 3, 4, 5, 6, 1, 'Hello World', 3.14, True)
```

In [ ]:
```
mixed_tuple * 3
```

Out[ ]:
```
(1,
 'Hello World',
 3.14,
 True,
 1,
 'Hello World',
 3.14,
 True,
 1,
 'Hello World',
 3.14,
 True)
```

In [ ]:
```
numbers *3
```

Out[ ]:
```
(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6)
```

In [ ]:
```
## Immutable Nature Of Tuples
## Tuples are immutable, meaning their elements cannot be changed once assigned.

lst=[1,2,3,4,5]
print(lst)

lst[1]="SANAD"
print(lst)
```

```
[1, 2, 3, 4, 5]
[1, 'SANAD', 3, 4, 5]
```

In [ ]:
```
numbers[1]="Singh"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-273-9c65f92b66ef> in <cell line: 1>()
----> 1 numbers[1]="Singh"

TypeError: 'tuple' object does not support item assignment
```

In [ ]:
```
numbers
```

Out[ ]:
```
(1, 2, 3, 4, 5, 6)
```

In [ ]:
```
## Tuple Methods
print(numbers.count(1))
print(numbers.index(3))
```

```
1
2
```

In [ ]:
```
## Packing and Unpacking tuple
## packing
packed_tuple=1,"Hello",3.14
print(packed_tuple)
```

```
(1, 'Hello', 3.14)
```

In [ ]:
```
##unpacking a tuple
a,b,c=packed_tuple

print(a)
print(b)
print(c)
```

```
1
Hello
3.14
```

In [ ]:
```python
## Unpacking with *
numbers=(1,2,3,4,5,6)
first,*middle,last=numbers
print(first)
print(middle)
print(last)
```

```
1
[2, 3, 4, 5]
6
```

In [ ]:
```python
## Nested Tuple
## Nested List
lst=[[1,2,3,4],[6,7,8,9],[1,"Hello",3.14,"c"]]
lst[0][0:3]
```

Out[ ]:
```
[1, 2, 3]
```

In [ ]:
```python
nested_tuple = ((1, 2, 3), ("a", "b", "c"), (True, False))

## access the elements inside a tuple
print(nested_tuple[0])
print(nested_tuple[1][2])
```

```
(1, 2, 3)
c
```

In [ ]:
```python
## iterating over nested tuples
for sub_tuple in nested_tuple:
    for item in sub_tuple:
        print(item,end=" ")
    print()
```

```
1 2 3
a b c
True False
```

KKB

- Tuples are versatile and useful in many real-world scenarios where an immutable and ordered collection of items is required.
- They are commonly used in data structures, function arguments and return values, and as dictionary keys.
- Understanding how to leverage tuples effectively can improve the efficiency and readability of your Python code.

Sets

- Sets are a built-in data type in Python used to store collections of unique items.
- They are unordered, meaning that the elements do not follow a specific order, and they do not allow duplicate elements.
- Sets are useful for membership tests, eliminating duplicate entries, and performing mathematical set operations like union, intersection, difference, and symmetric difference.

In [ ]:
```
##create a set
my_set={1,2,3,4,5}
print(my_set)
print(type(my_set))
```

```
{1, 2, 3, 4, 5}
<class 'set'>
```

In [ ]:
```
empty_set=set()
print(type(empty_set))
```

```
<class 'set'>
```

In [ ]:
```
my_set=set([1,2,3,4,5,6])
print(my_set)
```

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

In [ ]:
```
empty_set=set([1,2,3,6,5,4,5,6])
print(empty_set)
```

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

In [ ]:
```
## Basics Sets Operation
## Adiing and Removing Elements
my_set.add(7)
print(my_set)
my_set.add(7)
print(my_set)
```

```
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}
```

In [ ]:
```
## Remove the elements from a set
my_set.remove(3)
print(my_set)
```

```
{1, 2, 4, 5, 6, 7}
```

In [ ]:
```
my_set.remove(10)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-299-aa5e57d43179> in <cell line: 1>()
----> 1 my_set.remove(10)

KeyError: 10
```

In [ ]:
```
my_set.discard(11)
print(my_set)
```

```
{1, 2, 4, 5, 6, 7}
```

In [ ]:
```
## pop method
removed_element=my_set.pop()
print(removed_element)
print(my_set)
```

```
1
{2, 4, 5, 6, 7}
```

In [ ]:
```
## clear all the elements
my_set.clear()
print(my_set)
```

```
set()
```

In [ ]:
```
## Set Memebership test
my_set={1,2,3,4,5}
print(3 in my_set)
print(10 in my_set)
```

```
True
False
```

In [ ]:
```
## MAthematical Operation
set1={1,2,3,4,5,6}
set2={4,5,6,7,8,9}

### Union
union_set=set1.union(set2)
print(union_set)

## Intersection
intersection_set=set1.intersection(set2)
print(intersection_set)

set1.intersection_update(set2)
print(set1)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{4, 5, 6}
{4, 5, 6}
```

In [ ]:
```
set1={1,2,3,4,5,6}
set2={4,5,6,7,8,9}

## Difference
print(set1.difference(set2))
```

```
{1, 2, 3}
```

In [ ]:
```
set1
```

Out[ ]:
```
{1, 2, 3, 4, 5, 6}
```

In [ ]:
```
set2.difference(set1)
```

Out[ ]:
```
{7, 8, 9}
```

In [ ]:
```
## Symmetric Difference
set1.symmetric_difference(set2)
```

Out[ ]:
```
{1, 2, 3, 7, 8, 9}
```

In [ ]:
```
## Sets Methods
set1={1,2,3,4,5}
set2={3,4,5}

## is subset
print(set1.issubset(set2))

print(set1.issuperset(set2))
```

```
False
True
```

In [ ]:
```
lst=[1,2,2,3,4,4,5]
set(lst)
```

Out[ ]:
```
{1, 2, 3, 4, 5}
```

In [ ]:
```
### Counting Unique words in text

text="I an working a data science intern"
words=text.split()

## convert list of words to set to get unique words

unique_words=set(words)
print(unique_words)
print(len(unique_words))
```

```
{'a', 'I', 'science', 'working', 'intern', 'an', 'data'}
7
```

Key Points

- Sets are a powerful and flexible data type in Python that provide a way to store collections of unique elements.
- They support various operations such as union, intersection, difference, and symmetric difference, which are useful for mathematical computations.
- Understanding how to use sets and their associated methods can help you write more efficient and clean Python code, especially when dealing with unique collections and membership tests.

Introduction to Dictionaries

- Dictionaries are unordered collections of items.
- They store data in key-value pairs.
- Keys must be unique and immutable (e.g., strings, numbers, or tuples), while values can be of any type.

In [ ]:
```
## Creating Dictionaries
empty_dict={}
print(type(empty_dict))
```

```
<class 'dict'>
```

In [ ]:
```
empty_dict=dict()
empty_dict
```

Out[ ]:
```
{}
```

In [ ]:
```
student={"name":"sanad","age":28,"grade":12}
print(student)
print(type(student))
```

```
{'name': 'sanad', 'age': 28, 'grade': 12}
<class 'dict'>
```

In [ ]:
```
# Single key is always used
student={"name":"Sanad","age":30,"name":12}
print(student)
```

```
{'name': 12, 'age': 30}
```

In [ ]:
```
## accessing Dictionary Elements
student={"name":"sanad","age":28,"grade":"A"}
print(student)
```

```
{'name': 'sanad', 'age': 28, 'grade': 'A'}
```

In [ ]:
```
## Accessing Dictionary elements
print(student['grade'])
print(student['age'])
```

```
A
28
```

In [ ]:
```
## Accessing using get() method
print(student.get('grade'))
print(student.get('last_name'))
print(student.get('last_name',"Not Available"))
```

```
A
None
Not Available
```

- Modifying Dicitonary Elements
- Dictionary are mutable,so you can add, update or delete elements

In [ ]:
```
print(student)
```

```
{'name': 'sanad', 'age': 28, 'grade': 'A'}
```

In [ ]:
```
student["age"]=33  ##update value for the key
print(student)
student["address"]="Uttar Pradesh" ## added a new key and value
print(student)
```

```
{'name': 'sanad', 'age': 33, 'grade': 'A'}
{'name': 'sanad', 'age': 33, 'grade': 'A', 'address': 'Uttar Pradesh'}
```

In [ ]:
```
del student['grade'] ## delete key and value pair

print(student)
```

```
{'name': 'sanad', 'age': 33, 'address': 'Uttar Pradesh'}
```

In [ ]:
```
## Dictionary methods

keys=student.keys() ##get all the keys
print(keys)
```

```
dict_keys(['name', 'age', 'address'])
```

In [ ]:
```
values=student.values() ##get all values
print(values)
```

```
dict_values(['sanad', 33, 'Uttar Pradesh'])
```

In [ ]:
```
items=student.items() ##get all key value pairs
print(items)
```

```
dict_items([('name', 'sanad'), ('age', 33), ('address', 'Uttar Pradesh')])
```

In [ ]:
```
## shallow copy

student_copy=student
```

```
print(student)
print(student_copy)
```

```
{'name': 'sanad', 'age': 33, 'address': 'Uttar Pradesh'}
{'name': 'sanad', 'age': 33, 'address': 'Uttar Pradesh'}
```

In [ ]:
```
student["name"]="suraj"
print(student)
print(student_copy)
```

```
{'name': 'suraj', 'age': 33, 'address': 'Uttar Pradesh'}
{'name': 'suraj', 'age': 33, 'address': 'Uttar Pradesh'}
```

In [ ]:
```
student_copy1=student.copy() ## shallow copy

print(student_copy1)
print(student)
```

```
{'name': 'suraj', 'age': 33, 'address': 'Uttar Pradesh'}
{'name': 'suraj', 'age': 33, 'address': 'Uttar Pradesh'}
```

In [ ]:
```
student["name"]="Raj"
print(student_copy1)
print(student)
```

```
{'name': 'suraj', 'age': 33, 'address': 'Uttar Pradesh'}
{'name': 'Raj', 'age': 33, 'address': 'Uttar Pradesh'}
```

- Iterating Over Dictionaries
- You can use loops to iterate over dictionatries, keys,values,or items

In [ ]:
```
## Iterating over keys
for keys in student.keys():
    print(keys)
```

```
name
age
address
```

In [ ]:
```
## Iterate over values
for value in student.values():
    print(value)
```

```
Raj
33
Uttar Pradesh
```

In [ ]:
```
## Iterate over key value pairs
for key,value in student.items():
    print(f"{key}:{value}")
```

```
name:Raj
age:33
address:Uttar Pradesh
```

In [ ]:
```
## Nested Disctionaries
students={"student1":{"name":"Shams","age":32},"student2":{"name":"Peter","age":35}

print(students)
```

```
{'student1': {'name': 'Shams', 'age': 32}, 'student2': {'name': 'Peter', 'age': 3
5}}
```

In [ ]:
```
## Access nested dictionaries elementss
print(students["student2"]["name"])
```

```
print(students["student2"]["age"])
```

```
Peter
35
```

In [ ]:
```
print(students["student1"]["name"])
print(students["student1"]["age"])
```

```
Shams
32
```

In [ ]:
```
students.items()
```

Out[ ]:
```
dict_items([('student1', {'name': 'Shams', 'age': 32}), ('student2', {'name': 'Pet
er', 'age': 35})])
```

In [ ]:
```
## Iterating over nested dictionaries
for student_id,student_info in students.items():
    print(f"{student_id}:{student_info}")
    for key,value in student_info.items():
        print(f"{key}:{value}")
```

```
student1:{'name': 'Shams', 'age': 32}
name:Shams
age:32
student2:{'name': 'Peter', 'age': 35}
name:Peter
age:35
```

In [ ]:
```
## Dictionary Comphrehension
squares={x:x**2 for x in range(5)}
print(squares)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

In [ ]:
```
## Condition dictionary comprehension
evens={x:x**2 for x in range(10) if x%2==0}
print(evens)
```

```
{0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
```

Use a dictionary to count he frequency of elements in list

In [ ]:
```
numbers=[1,2,2,3,3,3,4,4,4,4]
frequency={}

for number in numbers:
    if number in frequency:
        frequency[number]+=1
    else:
        frequency[number]=1
print(frequency)
```

```
{1: 1, 2: 2, 3: 3, 4: 4}
```

Merge 2 dictionaries into one

In [ ]:
```
dict1={"a":1,"b":2}
dict2={"b":3,"c":4}
merged_dict={**dict1,**dict2}
print(merged_dict)
```

```
{'a': 1, 'b': 3, 'c': 4}
```

Key Points

- Dictionaries are powerful tools in Python for managing key-value pairs.
- They are used in a variety of real-world scenarios, such as counting word frequency, grouping data, storing configuration settings, managing phonebooks, tracking inventory, and caching results.
- Understanding how to leverage dictionaries effectively can greatly enhance the efficiency and readability of your code.

Introduction to Functions

Definition:

- A function is a block of code that performs a specific task.
- Functions help in organizing code, reusing code, and improving readability.

In [ ]:
```
## syntax
def function_name(parameters):
    """Docstring"""
    # Function body
    return expression
```

In [ ]:
```
## why functions?
num=24
if num%2==0:
    print("the number is even")
else:
    print("the number is odd")
```

```
the number is even
```

In [ ]:
```
def even_or_odd(num):
    """This function finds even or odd"""
    if num%2==0:
        print("the number is even")
    else:
        print("the number is odd")

## Call this function
even_or_odd(24)
```

```
the number is even
```

In [ ]:
```
def even_or_odd(num):
    """This function finds even or odd"""
    if num%2==0:
        print("the number is even")
    else:
        print("the number is odd")

## Call this function
even_or_odd(23)
```

```
the number is odd
```

In [ ]:
```
## function with multiple parameters

def add(a,b):
    return a+b
```

```
result=add(2,4)
print(result)
```

```
6
```

In [ ]:
```
## Default Parameters

def greet(name):
    print(f"Hello {name} welcome To the world of data")

greet("Sanad")
```

```
Hello Sanad welcome To the world of data
```

In [ ]:
```
def greet(name="Guest"):
    print(f"Hello {name} welcome To the world of data")

greet("Sanad")
```

```
Hello Sanad welcome To the world of data
```

In [ ]:
```
### Variable Length Arguments
## Positional And Keywords arguments

def numbers(*krish):
    for number in krish:
        print(number)
numbers(1,2,3,4,5,6,7,8,"Sanad")
```

```
1
2
3
4
5
6
7
8
Sanad
```

In [ ]:
```
## Positional arguments
def numbers(*args):
    for number in args:
        print(number)
numbers(1,2,3,4,5,6,7,8,"Sanad")
```

```
1
2
3
4
5
6
7
8
Sanad
```

In [ ]:
```
### Keywords Arguments

def details(**kwargs):
    for key,value in kwargs.items():
        print(f"{key}:{value}")
details(name="Sanad",age="28",country="India")
```

```
name:Sanad
age:28
country:India
```

In [ ]:
```
def details(*args,**kwargs):
    for val in args:
        print(f" Positional arument :{val}")

    for key,value in kwargs.items():
        print(f"{key}:{value}")

details(1,2,3,4,"Sooraj",name="Sanad",age="28",country="India")
```

```
 Positional arument :1
 Positional arument :2
 Positional arument :3
 Positional arument :4
 Positional arument :Sooraj
name:Sanad
age:28
country:India
```

In [ ]:
```
### Return statements
def multiply(a,b):
    return a*b
multiply(2,789)
```

Out[ ]:
```
1578
```

In [ ]:
```
### Return multiple parameters
def multiply(a,b):
    return a*b,a
multiply(2,549)
```

Out[ ]:
```
(1098, 2)
```

Lambda Functions in Python

- Lambda functions are small anonymous functions defined using the **lambda** keyword.
- They can have any number of arguments but only one expression.
- They are commonly used for short operations or as arguments to higher-order functions.

Syntax

lambda arguments: expression

In [ ]:
```
def addition(a,b):
    return a+b
addition(2,89)
```

Out[ ]:
```
91
```

In [ ]:
```
addition=lambda a,b:a+b
type(addition)
print(addition(5,6))
```

```
11
```

In [ ]:
```
def even(num):
    if num%2==0:
        return True
```

```
    else:
      return False
even(24)
```

Out[ ]:
```
True
```

In [ ]:
```
def even(num):
    if num%2==0:
        return True
    else:
      return False
even(13)
```

Out[ ]:
```
False
```

In [ ]:
```
even1=lambda num:num%2==0
even1(12)
```

Out[ ]:
```
True
```

In [ ]:
```
def addition(x,y,z):
    return x+y+z

addition(12,13,14)
```

Out[ ]:
```
39
```

In [ ]:
```
add=lambda x,y,z:x+y+z
add(12,13,14)
```

Out[ ]:
```
39
```

map()- applies a function to all items in a list

In [ ]:
```
numbers=[1,2,3,4,5,6]
def square(number):
    return number**2

square(6)
```

Out[ ]:
```
36
```

In [ ]:
```
numbers=[1,2,3,4,5,6]
def square(number):
    return number**2

square(3)
```

Out[ ]:
```
9
```

In [ ]:
```
list(map(lambda x:x**2,numbers))
```

Out[ ]:
```
[1, 4, 9, 16, 25, 36]
```

The map() Function in Python

- The map() function applies a given function to all items in an input list (or any other iterable) and returns a map object (an iterator).
- This is particularly useful for transforming data in a list comprehensively.

In [ ]:
```
def square(x):
    return x*x

square(98)
```

Out[ ]:
```
9604
```

In [ ]:
```
numbers=[1,2,3,4,5,6,7,8]

list(map(square,numbers))
```

Out[ ]:
```
[1, 4, 9, 16, 25, 36, 49, 64]
```

In [ ]:
```
## Lambda function with map
numbers=[1,2,3,4,5,6,7,8]
list(map(lambda x:x*x,numbers))
```

Out[ ]:
```
[1, 4, 9, 16, 25, 36, 49, 64]
```

In [ ]:
```
# Map multiple iterables

numbers1=[1,2,3]
numbers2=[4,5,6]

added_numbers=list(map(lambda x,y:x+y,numbers1,numbers2))
print(added_numbers)
```

```
[5, 7, 9]
```

map() to convert a list of strings to integers

Use map to convert strings to integers

In [ ]:
```
str_numbers = ['1', '2', '3', '4', '5']
int_numbers = list(map(int, str_numbers))

print(int_numbers)  # Output: [1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

In [ ]:
```
def get_name(person):
    return person['name']

people=[
    {'name':'Sanad','age':32},
    {'name':'Raj','age':33}
]
list(map(get_name,people))
```

Out[ ]:
```
['Sanad', 'Raj']
```

Key Points

- The map() function is a powerful tool for applying transformations to iterable data structures.
- It can be used with regular functions, lambda functions, and even multiple iterables, providing a versatile approach to data processing in Python.

- By understanding and utilizing map(), you can write more efficient and readable code.

The filter() Function in Python

- The filter() function constructs an iterator from elements of an iterable for which a function returns true.
- It is used to filter out items from a list (or any other iterable) based on a condition.

In [ ]:
```
def even(num):
    if num%2==0:
        return True
even(24)
```

Out[ ]:
```
True
```

In [ ]:
```
lst=[1,2,3,4,5,6,7,8,9,10,11,12]

list(filter(even,lst))
```

Out[ ]:
```
[2, 4, 6, 8, 10, 12]
```

Filter with a Lambda Function

In [ ]:
```
numbers=[1,2,3,4,5,6,7,8,9]
greater_than_five=list(filter(lambda x:x>5,numbers))
print(greater_than_five)
```

```
[6, 7, 8, 9]
```

Filter with a lambda function and multiple conditions

In [ ]:
```
numbers=[1,2,3,4,5,6,7,8,9]
even_and_greater_than_five=list(filter(lambda x:x>5 and x%2==0,numbers))
print(even_and_greater_than_five)
```

```
[6, 8]
```

Filter() to check if the age is greate than 25 in dictionaries

In [ ]:
```
people=[
    {'name':'raj','age':32},
    {'name':'siddhart','age':33},
    {'name':'saurabh','age':25}
]

def age_greater_than_25(person):
    return person['age']>25

list(filter(age_greater_than_25,people))
```

Out[ ]:
```
[{'name': 'raj', 'age': 32}, {'name': 'siddhart', 'age': 33}]
```

Key Pointers

- The filter() function is a powerful tool for creating iterators that filter items out of an iterable based on a function.
- It is commonly used for data cleaning, filtering objects, and removing unwanted elements from lists.
- By mastering filter(), you can write more concise and efficient code for processing and manipulating collections in Python.

Importing Modules in Python: Modules and Packages

In Python, modules and packages help organize and reuse code. Here's a comprehensive guide on how to import them.

In [ ]:
```
import math
math.sqrt(16)
```

Out[ ]:
```
4.0
```

In [ ]:
```
from math import sqrt,pi
print(sqrt(16))
print(sqrt(25))
print(pi)
```

```
4.0
5.0
3.141592653589793
```

In [ ]:
```
import numpy as np
np.array([1,2,3,4])
```

Out[ ]:
```
array([1, 2, 3, 4])
```

In [ ]:
```
from math import *
print(sqrt(16))
print(pi)
```

```
4.0
3.141592653589793
```

Some Maths Functions

In [ ]:
```
import random # Getting any random element from the list
seq=[1,3,5,7.1,9,10,15,14,18,19]
random.choice(seq)
```

Out[ ]:
```
1
```

In [ ]:
```
seq=["data","Tcs","infosys","hcl","sam","himanshu","kalay"] #any random element fro
random.choice(seq)
```

Out[ ]:
```
'sam'
```

In [ ]:
```
import random #got any random element from the range
start=508999456823654423654126 65
end=699998555555555666666699999
step=110000999789745862254
random.randrange(start,end,step)
```

Out[ ]:
```
519367473543944161460670943
```

In [ ]:
```
list1=[1,9,2,3,4,5,6,9,7,8,5,6,4,7] #getting suffelled elements of the list
random.shuffle(list1)
print(list1)
```

```
[4, 1, 6, 8, 5, 6, 9, 3, 7, 4, 9, 5, 2, 7]
```

In [ ]:
```
lower=99 #any random integer in-between the different variable
upper=130
random.randint(lower,upper)
```

Out[ ]:
```
99
```

In [ ]:
```
population=[20,21,23,24,25,26,27,28,29,30]  #got error because of float value, as s
k=5.5
random.sample(population,k)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-388-c33fd7418736> in <cell line: 3>()
      1 population=[20,21,23,24,25,26,27,28,29,30]  #got error because of float va
lue, as sample need only integer
      2 k=5.5
----> 3 random.sample(population,k)

/usr/lib/python3.10/random.py in sample(self, population, k, counts)
    481         if not 0 <= k <= n:
    482             raise ValueError("Sample larger than population or is negativ
e")
--> 483         result = [None] * k
    484         setsize = 21        # size of a small set minus size of an empty l
ist
    485         if k > 5:

TypeError: can't multiply sequence by non-int of type 'float'
```

In [ ]:
```
population=[20,21,23,24,25,26,27,28,29,30]  #overcome the error got 5 sample values
k=5
random.sample(population,k)
```

Out[ ]:
```
[30, 29, 20, 21, 26]
```

In [ ]:
```
population=[20,21,23,24,25,26,27,28,29,30] #got  7 sample values from the list
k=7
random.sample(population,k)
```

Out[ ]:
```
[30, 20, 24, 25, 26, 29, 21]
```

Number Representational function

- Ceil() - Upper value
- Floor() - Lower Value

In [ ]:
```
import math   #ceiling is upper value
number=8.10
print(math.ceil(number))
```

```
9
```

In [ ]:
```
print(math.floor(number)) #Lower value
```

```
8
```

fabs() or math.fabs()

In [ ]:
```
print(math.fabs(-8.10)) # converted the -ve float values to the +ve value
```

```
8.1
```

Factorial()

In [ ]:
```
print(math.factorial(5))
```

```
120
```

In [ ]:
```
print(math.factorial(-5)) #factorial of negative values are not possible

  #ValueError: factorial() not defined for negative values
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-395-f7dd80158980> in <cell line: 1>()
----> 1 print(math.factorial(-5)) #factorial of negative values are not possible
      2
      3   #ValueError: factorial() not defined for negative values

ValueError: factorial() not defined for negative values
```

fmod()

- returns x%y or always gives the remainder

In [ ]:
```
print(math.fmod(5,2))
```

```
1.0
```

In [ ]:
```
print(math.fmod(-5,2))
```

```
-1.0
```

In [ ]:
```
print(math.fmod(-5.2,2)) #negative value with mod
```

```
-1.2000000000000002
```

In [ ]:
```
print(math.fmod(5.2,3)) #positve value with Mod
```

```
2.2
```

fsum()

- sum of float values (list, tupple) # when list tupple is given

In [ ]:
```
numbers=[.1,.2,.3,.4,.5,.6,.7,.8,8.9]
print("Sum of ",numbers," is :",math.fsum(numbers))  #cumulative sum of all element
```

```
Sum of  [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 8.9]  is : 12.5
```

In [ ]:
```
print("Sum of 1 to 10 numbers is :",math.fsum(range(1,11))) #when list is not giver
```

```
Sum of 1 to 10 numbers is : 55.0
```

In [ ]:
```
math.e #exponential value
```

Out[ ]:
```
2.718281828459045
```

In [ ]:
```
math.pi #mantisa decimal point after decimal point.
```

Out[ ]:
```
3.141592653589793
```

In [ ]:
```
print(math.log(1))
```

```
0.0
```

In [ ]:
```
print(math.log(2)) #base e
```

```
0.6931471805599453
```

In [ ]:
```
print(math.log(64,2)) #base 2
```

```
6.0
```

In [ ]:
```
#log10()--> log with base 10
print(math.log10(1000))
```

```
3.0
```

In [ ]:
```
print(math.log(1000,10))
```

```
2.9999999999999996
```

In [ ]:
```
print(math.log(254,4))
```

```
3.994342343386083
```

In [ ]:
```
print(math.pow(5,4)) #power
```

```
625.0
```

In [ ]:
```
#sqrt()
print(math.sqrt(256)) #square root
```

```
16.0
```

Trignometric Functions

In [ ]:
```
print("Sin pi/2:",math.sin(90))  #result radians
```

```
Sin pi/2: 0.8939966636005579
```

In [ ]:
```
print("Sin pi/2:",math.sin(math.pi/2)) #result degree
```

```
Sin pi/2: 1.0
```

In [ ]:
```
print("tan pi/4:",math.tan(math.pi/4))
```

```
tan pi/4: 0.9999999999999999
```

In [ ]:
```
print(math.degrees(1.57))   #1.57 radians
```

```
89.95437383553924
```

In [ ]:
```
print(math.radians(89.95437383553924))
```

```
1.57
```

Number Datatypes

- int--> positive or negative whole number with no decimal point.
- float---> floating point numbers
- complex --> (x+yj) where x and y -->real values and j --> imaginary number
- long ---> NO
- double --->NO

In [ ]:
```
#isinstance() #checking data type
a=1234
isinstance(a,int)
```

Out[ ]:
```
True
```

In [ ]:
```
a=1234
isinstance(a,float)
```

Out[ ]:
```
False
```

In [ ]:
```
#Number system-->   prefixed
#0B or 0b --> binary ---->binary 2
#0O or 0o---> octal  ----->8
#0x or 0X ---> Hexadecimal --->16   A=10,B=11,C=12,D=13,E=15
```

In [ ]:
```
print(0b1110001)
```

```
113
```

In [ ]:
```
bin(113)
```

Out[ ]:
```
'0b1110001'
```

In [ ]:
```
x=0o15
print(x)
```

```
13
```

In [ ]:
```
oct(13)
```

Out[ ]:
```
'0o15'
```

In [ ]:
```
x=0xBF
print(x)
```

```
191
```

In [ ]:
```
x=112.34
print(type(x))
print(int(x))
```

```
<class 'float'>
112
```

In [ ]:
```
x=1234
float(x)
```

Out[ ]:
```
1234.0
```

In [ ]:
```
x=123
complex(x)
```

Out[ ]:
```
(123+0j)
```

In [ ]:
```
x=12.34
complex(x)
```

Out[ ]:
```
(12.34+0j)
```

In [ ]:
```
x=12.34
y=56
z=complex(x,y)
z
```

Out[ ]:
```
(12.34+56j)
```

In [ ]:
```
from math import pi
print(pi)
```

```
3.141592653589793
```

In [ ]:
```
import math  as m
m.pi
```

Out[ ]:
```
3.141592653589793
```

In [ ]:
```
from fractions import Fraction as fr
```

In [ ]:
```
fr(2.2) #higgest value give
```

Out[ ]:
```
Fraction(2476979795053773, 1125899906842624)
```

In [ ]:
```
fr('2.2')
```

Out[ ]:
```
Fraction(11, 5)
```

In [ ]:
```
fr('2.5')
```

Out[ ]:
```
Fraction(5, 2)
```

In [ ]:
```
fr(1,3)+fr(3,1)
```

Out[ ]:
```
Fraction(10, 3)
```

In [ ]:
```
fr(1,3)-fr(3,1)
```

Out[ ]:
```
Fraction(-8, 3)
```

In [ ]:
```
fr(1,6)*fr(6,1)
```

Out[ ]:
```
Fraction(1, 1)
```

In [ ]:
```
fr(1,6)/fr(6,1)
```

Out[ ]:
```
Fraction(1, 36)
```

In [ ]:
```
n=9
bin(n)
```

Out[ ]:
```
'0b1001'
```

In [ ]:
```
n.bit_length()
```

Out[ ]:
```
4
```

In [ ]:
```
num1=13 #maximum value within the variable
num2=26
num3=33
max(num1,num2,num3)
```

Out[ ]:
```
33
```

In [ ]:
```
min(num1,num2,num3)
```

Out[ ]:
```
13
```

In [ ]:
```
num=12.345567
n=3
round(num,n)
```

Out[ ]:
```
12.346
```

In [ ]:
```
math.trunc(num)
```

Out[ ]:
```
12
```

What is ooop?

- Oops stands for Object-Oriented Programming System
- In computer science their are various proggraming launguage abailable for instructing systems or computing devices.
- oops is one of the fundamental concept for every computer launguage to provide a clear modular sturcture and effective way of solving problems.

What is ooop?

- Obeject oriented programiing is a programing paradigm, that deals with various fundamental of its concepts
- After the procedural laungage revolution, the oops cooncept has become an essential part of our programing world to achieve better productivity, flexiblity, userfriendlyness, rapid code management.

What is class?

- Class is nothing but, buleprint, templete,plan, design for an object. Class is a virtual entity.

What is Object?

- Object is a physical instance of a class

Defining a class?

-class classname:

"""documentation string

constructor

statement1.....

statementN"""

In [ ]:
```
class student:
  """we can get college student details"""
  def __init__(self,sid,sname,smobileno):
    self.sid=sid
    self.sname=sname
    self.smobileno=smobileno
    print("construction execution")
  def talk (self):
    print("method execution")
    print("hello my id is:", self.sid)
    print("hello my name is :",self.sname)
    print("hello my mobileno:",self.smobileno)
obj=student(2453, "sanad",1234567897)
print(student.__doc__)
obj.talk()
```

```
construction execution
we can get college student details
method execution
hello my id is: 2453
hello my name is : sanad
hello my mobileno: 1234567897
```

In [ ]:
```
a= 1
print(type(a))
print(type("Sanad Singh"))
```

```
<class 'int'>
<class 'str'>
```

In [ ]:
```
class test:
  pass
a=test()
print(type(a))
```

```
<class '__main__.test'>
```

In [ ]:
```
class dataloves: # Class define
  def welcome(self): #constructor
    print("welcome to the hollow earth") #statement
obj=dataloves() #object calling
obj.welcome()
```

```
welcome to the hollow earth
```

In [ ]:
```
obj2=dataloves()
obj2.welcome()
```

```
welcome to the hollow earth
```

In [ ]:
```
a=1,
b=2
print(a,b)
```

```
(1,) 2
```

In [ ]:
```
class dataloves:
  def __init__(self, phone_number,email_id,student_id):
    self.phone_number=phone_number
    self.email_id=email_id
    self.student_id=student_id
    print("Student Details")
  def talk(self):
    print("Method execution")
obj=dataloves(4256489,"sand@hotmail.com",235)
obj.talk()
```

```
Student Details
Method execution
```

write a programm to get student details

In [ ]:
```
class student:
  "Here we got the student details"
  def __init__(self,sid,sname,snumber):
    self.sid=sid
    self.sname=sname
    self.snumber=snumber
  def talk(self):
    print("Student Details")
    print("Student Id:", self.sid)
    print("Student Name:", self.sname)
    print("Student Name:", self.snumber)
obj=student(987654321,"himanshu",987654321)
print(student.__doc__)
obj.talk()
```

```
Here we got the student details
Student Details
Student Id: 987654321
Student Name: himanshu
Student Name: 987654321
```

In [ ]:
```
# write a programm to get empployee  details using OOPS
class emp:
  def __init__(self,empname,empage,empcompany,empsalary,emprole):
    self.empname=empname
    self.empage=empage
    self.empcompany=empcompany
    self.empsalary=empsalary
    self.emprole=emprole
  def talk(self):
    print("method exection")
    print("hello my name is:", self.empname)
    print("hello my age is:",self.empname)
    print("hello my company is:",self.empcompany)
    print("hello my salary is:",self.empsalary)
    print("hello my role is:",self.emprole)
obj=emp("sanad",29,"deeshaw",52000,"datascientist")
print(emp.__doc__)
obj.talk()
```

```
None
method exection
hello my name is: sanad
hello my age is: sanad
hello my company is: deeshaw
hello my salary is: 52000
hello my role is: datascientist
```

Write a programm to get details of multiple employee using OOPS.

In [ ]:
```
class emp:
  "we can get employee data using OOPs"
  def __init__(self,empname,empage,empcompany,empsalary,emprole):
    self.empname=empname
    self.empage=empage
    self.empcompany=empcompany
    self.empsalary=empsalary
    self.emprole=emprole
  def talk(self):
    print("method exection")
    print("hello my name is:", self.empname)
    print("hello my age is:",self.empage)
    print("hello my company is:",self.empcompany)
    print("hello my salary is:",self.empsalary)
    print("hello my role is:",self.emprole)
obj=emp("sanad",29,"deeshaw",52000,"datascientist")
obj1=emp("himanshu",30,"deeshaw",60000,"datascientist")
print(emp.__doc__)
obj.talk()
obj1.talk()
```

```
we can get employee data using OOPs
method exection
hello my name is: sanad
hello my age is: 29
hello my company is: deeshaw
hello my salary is: 52000
hello my role is: datascientist
method exection
hello my name is: himanshu
hello my age is: 30
hello my company is: deeshaw
hello my salary is: 60000
hello my role is: datascientist
```

In [ ]:
```
class emp:
  "we can get employee data using OOPs"
  def __init__(self,empname,empage,empcompany,empsalary,emprole):
    self.empname=empname
    self.empage=empage
    self.empcompany=empcompany
    self.empsalary=empsalary
    self.emprole=emprole
  def talk(self):
    print("hello my name is:", self.empname)
    print("hello my age is:",self.empage)
    print("hello my company is:",self.empcompany)
    print("hello my salary is:",self.empsalary)
    print("hello my role is:",self.emprole)
obj=emp("sanad",29,"deeshaw",52000,"datascientist")
print(emp.__doc__)
obj.talk()
```

```
we can get employee data using OOPs
hello my name is: sanad
hello my age is: 29
hello my company is: deeshaw
hello my salary is: 52000
hello my role is: datascientist
```

In [ ]:
```
class emp:
  "we can get employee data using OOPs"
  def __init__(self,empname,empage,empcompany,empsalary,emprole):
    self.empname=empname
    self.empage=empage
    self.empcompany=empcompany
    self.empsalary=empsalary
    self.emprole=emprole
  def talk(self):
    print("hello my name is:", self.empname)
    print("hello my age is:",self.empname)
    print("hello my company is:",self.empcompany)
    print("hello my salary is:",self.empsalary)
    print("hello my role is:",self.emprole)

obj=emp("sanad",29,"deeshaw",52000,"datascientist")

obj1=emp("himanshu",30,"deeshaw",60000,"datascientist")

print(emp.__doc__)

obj.talk()
print("------------------------------------------------------------")
```

```
obj1.talk()
print("------------------------------------------------------------")
```

```
we can get employee data using OOPs
hello my name is: sanad
hello my age is: sanad
hello my company is: deeshaw
hello my salary is: 52000
hello my role is: datascientist
------------------------------------------------------------
hello my name is: himanshu
hello my age is: himanshu
hello my company is: deeshaw
hello my salary is: 60000
hello my role is: datascientist
------------------------------------------------------------
```

In [ ]:
```
print(help(emp))
```

```
Help on class emp in module __main__:

class emp(builtins.object)
 |  emp(empname, empage, empcompany, empsalary, emprole)
 |
 |  we can get employee data using OOPs
 |
 |  Methods defined here:
 |
 |  __init__(self, empname, empage, empcompany, empsalary, emprole)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  talk(self)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

None
```

In [ ]:
```
class emp:
  def __init__(self):
    print(id(self))
obj=emp()
print(id(obj))
```

```
140498111619392
140498111619392
```

In [ ]:
```
class emp:
  "we can get employee data using OOPs"
  def __init__(self,empname,empage,empcompany,empsalary,emprole):
    self.empname="sanad"
    self.empage=29
    self.empcompany="deeshaw"
    self.empsalary=520000
    self.emprole="datascientist"
    print("constructor execution")
  def talk(self):
    print("method exection")
    print("hello my name is:", self.empname)
```

```
    print("hello my age is:",self.empage)
    print("hello my company is:",self.empcompany)
    print("hello my salary is:",self.empsalary)
    print("hello my role is:",self.emprole)
obj=emp("sanad",29,"deeshaw",52000,"datascientist")
print(emp.__doc__)
obj.talk()
```

```
constructor execution
we can get employee data using OOPs
method exection
hello my name is: sanad
hello my age is: 29
hello my company is: deeshaw
hello my salary is: 520000
hello my role is: datascientist
```

Constructor

- Whenever we are creating object, constructor will be executed automatically and we are not required to call expcitelly.
- The main objective of the constructor is to declair and initilize variable.
- For every object constructor will be executed only once. This will be called as default constructor.

What is Self?

- Self is an implicite variable which is always provided by the PVM 'python virtual machine.
- Self is always pointing to the current onject.
- For every constructor an instance method, the first argument should be self.

In [ ]:
```
class test:
  def __init__(self):
    print("constructor execution")
t=test()
```

```
constructor execution
```

In [ ]:
```
class test:
  def __init__(self):
    self.x=10
    self.y=20
  def demo(self):
    print(self.x)
    print(self.y)
obj=test()
obj.demo()
```

```
10
20
```

In [ ]:
```
class test:
  def __init__(self):
    self.x=10
    self.y=20
  def demo(self):
```

```
    print(self.x)
    print(self.y)
t=test()
```

In [ ]:
```
class test:  #if two constructor, or two method but different argument it will be k
  def __init__(self):
    print("no argument")
  def __init__(self,x):
    print("one argument")
t1=test(10)
```

```
one argument
```

Printing multiple candidates details

In [ ]:
```
class student:
  cname="dataloves"
  def __init__(self,sid,sname,sage):
    self.sid=sid
    self.sname=sname
    self.sage=sage
  def display(self):
    print("hello my id is:", self.sid)
    print("hello my name is :",self.sname)
    print("hello my age is:",self.sage)
    print("hello my college name is:",self.cname)
obj=student(2453, "sanad",29)
obj2=student(111,"himanshu",65)
obj3=student(564,"Raj",69)
print("data1")
obj.display()
print("data2")
obj2.display()
print("data3")
obj3.display()
```

```
data1
hello my id is: 2453
hello my name is : sanad
hello my age is: 29
hello my college name is: dataloves
data2
hello my id is: 111
hello my name is : himanshu
hello my age is: 65
hello my college name is: dataloves
data3
hello my id is: 564
hello my name is : Raj
hello my age is: 69
hello my college name is: dataloves
```

In [ ]:
```
class student:
  cname="dataloves" #statice variable
  def __init__(self,sid,sname,sage): #constructor parametrized
    self.sid=sid #instance variable
    self.sname=sname
    self.sage=sage
  def display (self): #instance method
    print("hello my id is:", self.sid) #instance varibale
    print("hello my name is :",self.sname)
    print("hello my age is:",self.sage)
```

```
  @classmethod #decorator
  def getCollegename(cls):
    print("college name is:", cls.cname)
    #print("hello my college name is:",self.cname)
    #use of statice variable by self, but we are not reccomended to do this.
obj=student(2453, "sanad",29)
obj2=student(111,"himanshu",65)
obj3=student(564,"Raj",69)
student.getCollegename()
print("data1")
obj.display()
print("data2")
obj2.display()
print("data3")
obj3.display()
```

```
college name is: dataloves
data1
hello my id is: 2453
hello my name is : sanad
hello my age is: 29
data2
hello my id is: 111
hello my name is : himanshu
hello my age is: 65
data3
hello my id is: 564
hello my name is : Raj
hello my age is: 69
```

In [ ]:

```
class student:
  cname="dataloves" #statice variable
  def __init__(self,sid,sname,sage): #constructor parametrized
    self.sid=sid #instance variable
    self.sname=sname
    self.sage=sage
  def display (self): #instance method
    print("hello my id is:", self.sid) #instance varibale
    print("hello my name is :",self.sname)
    print("hello my age is:",self.sage)
  @classmethod #decorator
  def getCollegename(cls):
    print("college name is:", cls.cname)
  def findavg(x,y):
    print("Average is:",(x+y)/2 )
    #print("hello my college name is:",self.cname) #use of statice variable by self
obj=student(2453, "sanad",29)
obj2=student(111,"himanshu",65)
obj3=student(564,"Raj",69)
student.getCollegename()
student.findavg(10,5)
print("data1")
obj.display()
print("data2")
obj2.display()
print("data3")
obj3.display()
```

```
college name is: dataloves
Average is: 7.5
data1
hello my id is: 2453
hello my name is : sanad
hello my age is: 29
data2
hello my id is: 111
hello my name is : himanshu
hello my age is: 65
data3
hello my id is: 564
hello my name is : Raj
hello my age is: 69
```

What is python constructor?

- In programm construction, every statement like data attributes, class members, methods and other objects must be present within the class boundary.
- A constructor is a special type of method that help to initilize a newly created object.
- Depending on constructor (init method) we can pass ant no of the arguments by creating the class objects.
- Their are two type of constructor available in python programm in order to initlize instance member of the class.
- Non Parameterized and Parameterized constructors.

In [ ]:
```
class dataloves: #Example of Non Parametrized Constructor

  community=""
  def __init__(self):
    self.community="welcome to data trainee"
  def display(self):
    print(self.community)
obj=dataloves()
obj.display()
```

```
welcome to data trainee
```

In [ ]:
```
# Example for Parametrized Constructor

class fruits:
  def __init__(self,fname,fcolor):
    self.fname=fname
    self.fcolor=fcolor
  def display(self):
    print("fruit name:",self.fname)
    print("fruit color",self.fcolor)
fruit=fruits("orange","orange")
fruit2=fruits("apple","red")
fruit3=fruits("banana","yellow")
fruit4=fruits("Pear","green")
fruit.display()
fruit2.display()
fruit3.display()
fruit4.display()
```

```
fruit name: orange
fruit color orange
fruit name: apple
fruit color red
fruit name: banana
fruit color yellow
fruit name: Pear
fruit color green
```

In [ ]:
```python
class fruits:
  total=0
  def __init__(self,fname,fcolor):
    self.fname=fname
    self.fcolor=fcolor
    fruits.total+=1
  def display(self):
    print("fruit name:",self.fname)
    print("fruit color",self.fcolor)
fruit=fruits("orange","orange")
fruit2=fruits("apple","red")
fruit3=fruits("banana","yellow")
fruit4=fruits("Pear","green")
fruit.display()
fruit2.display()
fruit3.display()
fruit4.display()
print(fruits.total) #total number of fruits
```

```
fruit name: orange
fruit color orange
fruit name: apple
fruit color red
fruit name: banana
fruit color yellow
fruit name: Pear
fruit color green
4
```

Arithmetic Operations using Oops

In [ ]:
```python
class calculation:
  def __init__(self,a,b):
    self.a=a
    self.b=b
  def add(self):
    print("addidtion:",self.a+self.b)
  def sub(self):
    print("sub:",self.a-self.b)
  def multiplication(self):
    print("multiplication:",self.a*self.b)
  def division(self):
    print("division:",self.a/self.b)
  def floorDivision(self):
    print("floordivision:",self.a//self.b)
  def mod(self):
    print("modulous:",self.a%self.b)
obj=calculation(a=int(input("enter the number")),b=int(input("enter the number")))
obj.add()
obj.sub()
obj.multiplication()
obj.division()
```

```
obj.floorDivision()
obj.mod()
```

```
enter the number10
enter the number5
addidtion: 15
sub: 5
multiplication: 50
division: 2.0
floordivision: 2
modulous: 0
```

In [ ]:
```
class calculation:
  def __init__(self,a,b):
    self.a=a
    self.b=b
  def add(self):
    print("addidtion:",self.a+self.b)
  def sub(self):
    print("sub:",self.a-self.b)
  def multiplication(self):
    print("multiplication:",self.a*self.b)
  def division(self):
    print("division:",self.a/self.b)
  def floorDivision(self):
    print("floordivision:",self.a//self.b)
  def mod(self):
    print("modulous:",self.a%self.b)
p=int(input("enter the number"))
q=int(input("enter the number"))
obj=calculation(p,q)
obj.add()
obj.sub()
obj.multiplication()
obj.division()
obj.floorDivision()
obj.mod()
```

```
enter the number5
enter the number9
addidtion: 14
sub: -4
multiplication: 45
division: 0.5555555555555556
floordivision: 0
modulous: 5
```

Accessing instance attribiute

- In python we can access the object, variable using the dot operator, for calling the instance methodd in a programm you should create an object from the class that will provide accibility of class member, an attributes with the help of the dot operator

In [ ]:
```
class community:
  def __init__(sanad,name,year):
    sanad.name=name
    sanad.year=year
  def demo(sanad):
    print("hello good aternoon")
obj=community("datalove is a community","year of establismnet was 2019")
obj.demo()
```

```
hello good aternoon
```

In [ ]:
```
class community:
  def __init__(sanad,name,year):
    sanad.name=name
    sanad.year=year
  def demo(sanad):
    print("hello good aternoon")
obj=community("data is a community","year of establismnet was 1990")
obj.demo()
print("welcome to",obj.name,"and",obj.year) #dot operator
```

```
hello good aternoon
welcome to data is a community and year of establismnet was 1990
```

Accessing attribute inbuilt class function

- getattr(obj,name)- it allow us to access an attriute of an object
- setattr(obj,name)- This method use to set an attruibute if it created when it dosnt exist.
- hasattr(obj,name)-to check the attribute exist or not
- delattr(obj,name)-to delete an attribute

In [ ]:
```
class dog:
  name="dommy"
  age=8
  def show(self):
    print(self.name)
    print(self.age)
obj=dog()
print(getattr(obj,'height'))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-480-f39a7d263c7d> in <cell line: 8>()
      6     print(self.age)
      7 obj=dog()
----> 8 print(getattr(obj,'height'))

AttributeError: 'dog' object has no attribute 'height'
```

In [ ]:
```
class dog:
  name="dommy"
  age=8
  def show(self):
    print(self.name)
    print(self.age)
obj=dog()
print(getattr(obj,'name'))
```

```
dommy
```

In [ ]:
```
class dog:
  name="dommy"
  age=8
  def show(self):
    print(self.name)
    print(self.age)
obj=dog()
print(hasattr(obj,'height'))
```

```
False
```

In [ ]:
```
class dog:
  name="dommy"
  age=8
  def show(self):
    print(self.name)
    print(self.age)
obj=dog()
setattr(obj,'height',175)
```

In [ ]:
```
print(hasattr(obj,"height"))
```

```
True
```

In [ ]:
```
print(getattr(obj,"height"))
```

```
175
```

In [ ]:
```
print(hasattr(obj,"age"))
```

```
True
```

In [ ]:
```
delattr(dog,"age") #only class will be able to perform del operation.
```

In [ ]:
```
print(hasattr(obj,"age"))
```

```
False
```

Built-in class attributes

- pythob built-in class attributes give us some info about the class.
- Every class in python can retain below an attribute an access by using the dot operator.
- **dict** - to holding the class information
- **doc** - to give the class documentation string
- **name** - to give the class name
- **module**-it provide the module name in which the class is defined
- **bases**-to give the bases(in other words object information)

In [ ]:
```
class colors:
  "this  is a sample class called colors"
  colorCount=0
  def __init__(self,red,yellow):
    self.red="apple"
    self.yellow="lemon"
    colors.colorCount+=1
  def display(self):
    print("red color fruit is:",self.red)
    print("yellow color fruit is:",self.yellow)
obj=colors("apple","lemon")
obj.display()
print(colors.colorCount) #only one time constructor is working, constructor only co
```

```
red color fruit is: apple
yellow color fruit is: lemon
1
```

In [ ]:
```
class colors:
  "this  is a sample class called colors" #documentation string
  colorCount=0 #global Variable
  def __init__(self,red,yellow): #constructor
```

```
    self.red="apple" #instance variable
    self.yellow="lemon"#instance variable
    colors.colorCount+=1 #instance variable
  def display(self): #Instance Variable
    print("red color fruit is:",self.red)
    print("yellow color fruit is:",self.yellow)

print("colors.__doc__",colors.__doc__)
```

```
colors.__doc__ this  is a sample class called colors
```

In [ ]:
```
class colors:
  "this  is a sample class called colors" #documentation string
  colorCount=0 #global Variable
  def __init__(self,red,yellow): #constructor
    self.red="apple" #instance variable
    self.yellow="lemon"#instance variable
    colors.colorCount+=1 #instance variable
  def display(self): #Instance Variable
    print("red color fruit is:",self.red)
    print("yellow color fruit is:",self.yellow)

print("colors.__name__",colors.__name__)  #builtin attributes
print("colors.__module__",colors.__module__)
print("colors.__bases__",colors.__bases__)
print("colors.__dict__",colors.__dict__)
```

```
colors.__name__ colors
colors.__module__ __main__
colors.__bases__ (<class 'object'>,)
colors.__dict__ {'__module__': '__main__', '__doc__': 'this  is a sample class cal
led colors', 'colorCount': 0, '__init__': <function colors.__init__ at 0x7fc8440f6
680>, 'display': <function colors.display at 0x7fc8440f5e10>, '__dict__': <attribu
te '__dict__' of 'colors' objects>, '__weakref__': <attribute '__weakref__' of 'co
lors' objects>}
```

In [ ]:
```
class employee:
  def __init__(self,ename,eId,esalary):
    self.ename=ename
    self.eId=eId
    self.esalary=esalary
  def info(self):
    self.eloc="banglore"
    print("employee name:",self.ename)  #No need to use this, as we need data in di
    print("employee Id:",self.eId)
    print("employee salary:",self.esalary)
obj=employee("sanad",265,415467)
obj.info()
```

```
employee name: sanad
employee Id: 265
employee salary: 415467
```

In [ ]:
```
class employee:
  def __init__(self,ename,eId,esalary):
    self.ename=ename
    self.eId=eId
    self.esalary=esalary
  def info(self):
    self.eloc="banglore"
obj=employee("sanad",265,415467)
obj.info()
print(obj.__dict__)
```

```
{'ename': 'sanad', 'eId': 265, 'esalary': 415467, 'eloc': 'banglore'}
```

In [ ]:
```
class employee:
  def __init__(self,ename,eId):
    self.ename=ename
    self.eId=eId
  def info(self):
    self.eSal=60000
obj=employee("sanad",265)
obj.info()
obj.eloc="banglore" #new addition in dictionary due to mutability
print(obj.__dict__)
```

```
{'ename': 'sanad', 'eId': 265, 'eSal': 60000, 'eloc': 'banglore'}
```

In [ ]:
```
class student:
  def __init__(self):
    self.a=10
    self.b=20
    self.c=30
  def m1(self):
    self.d=40
  def m2(self):
    self.e=50
obj=student()
obj.m1()
obj1=student()
obj1.m2()
obj1.p=200
obj1.q=300
print(obj.__dict__)
print(obj1.__dict__)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
{'a': 10, 'b': 20, 'c': 30, 'e': 50, 'p': 200, 'q': 300}
```

In [ ]:
```
class test:
  def __init__(self):
    self.a=10
    self.b=20
    self.c=30
  def delete(self):
    del self.b
    del self.c
obj=test()
print(obj.__dict__)
```

```
{'a': 10, 'b': 20, 'c': 30}
```

In [ ]:
```
class test:
  def __init__(self):
    self.a=10
    self.b=20
    self.c=30
  def delete(self):
    del self.b
    del self.c
obj=test()
obj.delete()
print(obj.__dict__)
```

```
{'a': 10}
```

In [ ]:
```
class test:
  def __init__(self):
    self.a=10
    self.b=20
```

```
    self.c=30
  def delete(self):
    del self
obj=test()
obj.delete()
print(obj.__dict__)
```

```
{'a': 10, 'b': 20, 'c': 30}
```

In [ ]:
```
class test:
  def __init__(self):
    self.a=10
    self.b=20
    self.c=30
obj=test()
obj1=test()
del obj.a
del obj1.b
print(obj.__dict__)
print(obj1.__dict__)
```

```
{'b': 20, 'c': 30}
{'a': 10, 'c': 30}
```

In [ ]:
```
class test:
  def __init__(self):
    self.a=10
    self.b=20
obj1=test()
obj2=test()
obj1.a=888
obj2.b=999
print(obj1.a,obj2.b)
print(obj2.a,obj2.b)
```

```
888 999
10 999
```

In [ ]:
```
class test:
  a=10
  def __init__(self):
    print("inside constructor")
    print(test.a)
    print(self.a)
  def m1(self):
    print("inside instance Method")
    print(test.a)
    print(self.a)
obj=test()
obj.m1()
```

```
inside constructor
10
10
inside instance Method
10
10
```

In [ ]:
```
class test:
  a=10
  def __init__(self):
    print("inside constructor")
    print(test.a)
    print(self.a)
  def m1(self):
```

```
    print("inside instance Method")
    print(test.a)
    print(self.a)
  def m2(cls):
    print("inside class method")
    print(test.a)
    print(cls.a)
  @staticmethod
  def m3():
    print("inside static method")
    print(test.a)
obj=test()
obj.m1()
obj.m2()
obj.m3()
print("inside from the class")
print(test.a)
print(obj.a)
```

```
inside constructor
10
10
inside instance Method
10
10
inside class method
10
10
inside static method
10
inside from the class
10
10
```

In [ ]:
```
class test:
  a=10
  def __init__(self):
    test.a=20
obj=test()
print(test.a)
print(obj.a) #case1
```

```
20
20
```

In [ ]:
```
class test:
  a=10
  def __init__(self):
    test.a=20
  @classmethod
  def m1(cls):
    cls.a=30
    test.a=40
obj=test()
obj.m1()
print(test.a)
print(obj.a) #case2
```

```
40
40
```

In [ ]:
```
class test:
  a=10
  def __init__(self):
    test.a=20
```

```
  @classmethod
  def m1(cls):
    cls.a=30
    test.a=40
  @staticmethod
  def m2():
    test.a=50
obj=test()
obj.m1()
obj.m2()
test.a=60
print(test.a)
print(obj.a) #case3
```

```
60
60
```

In [ ]:
```
class test:
  def m1(self):
    a=100
    print(a)
  def m2(self):
    b=200
    print(b)
obj=test()
obj.m1()
obj.m2()
```

```
100
200
```

In [ ]:
```
class test:
  def m1(self):
    a=100
    print(a)
  def m2(self):
    b=200
    print(a,b) #name error
obj=test()
obj.m1()
obj.m2()
```

```
100
(1,) 200
```

In [ ]:
```
x=100
class test:
  def m1(self):
    print(x)
  def m2(self):
    print(x)
obj=test()
obj.m1()
obj.m2()
```

```
100
100
```

In [ ]:
```
x=100
class test:
  x=777
  def m1(self):
    x=888
    print(x)
  def m2(self):
```

```
    print(x)
    print(test.x)
obj=test()
obj.m1()
```

```
888
```

In [ ]:
```
class animals:
  leg=4
  @classmethod
  def walk(cls,name):
    cls.name=name
    print(cls.name,"walk with", animals.leg, "legs")
animals.walk("dog")
animals.walk("cat")
```

```
dog walk with 4 legs
cat walk with 4 legs
```

In [ ]:
```
class employee:
  cname='Infosys' #static variable
  #print(cname)
  def _init_(self,eid,ename,esale,eloc): #constructor with parameter
    self.eid=eid #instance variable
    self.ename=ename #instance variable
    self.esale=esale
    self.eloc=eloc
  def display(self): #instance method
    print('Hello my id is=',self.eid)
    print('my name is=',self.ename)
    print('my salary is=',self.esale)
    print('my location is=',self.eloc)
    #print('my company name is=',self.cname)
  @classmethod
  def get_company_name(cls): #class method
    print('Hello my company name is=',cls.cname)   #class level variable
obj=employee(5426,'Honey', 45000,'Kanpur')
obj1=employee(4236,'Sooraj',55000,"Noeda")
employee.get_company_name()
obj.display() #object by calling
obj1.display()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-511-291a96391270> in <cell line: 18>()
     16   def get_company_name(cls): #class method
     17     print('Hello my company name is=',cls.cname)   #class level variable
---> 18 obj=employee(5426,'Honey', 45000,'Kanpur')
     19 obj1=employee(4236,'Sooraj',55000,"Noeda")
     20 employee.get_company_name()

TypeError: employee() takes no arguments
```

What is Inheriterance?

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class"""

Type of Inheriterance

- Single level inheriterance
- multilevel inheriterance
- multiple inheriterance
- Hierarchical
- Hybrid

In [ ]:
```
#Single level inheriterance:
class parent:                    #single level inheriterance syntax
  pass
class child(parent):
  pass
```

In [ ]:
```
#multilevel inheriterance #synatx
class a:    #grandfather
  pass
class b(a): #father
  pass
class c(b): #child
  pass
```

In [ ]:
```
#multiple Inheriterance : One child class with multiple parent class called multipl
class a: #father
  pass
class b: #father
  pass
class c(a,b): #child
  pass
```

In [ ]:
```
#Hierarchical Inheriterance : One father with multiple childs.
class a: #father
  pass
class b(a): #child
  pass
class c(a): #child
  pass
```

- Hybrid Inheriterance: combination of multiple with hierarchial inheriterance.
- We cannot define syntex for Hybrid inheriterance.

In [ ]:
```
class a: #example for single level inheriterance
  def m1(self):
    print("m1 method")
class b(a):
  def m2(self):
    print("m2 method")
obj=b()
obj.m1()
obj.m2()
```

```
m1 method
m2 method
```

In [ ]:
```
#example for multi level inheriterance
class a:
  def m1(self):
    print("m1 method")
```

```
class b(a):
  def m2(self):
    print("m2 method")
class c(b):
  def m3(self):
    print("m3 method")
obj=c() #always current class object will be called
obj.m1()
obj.m2()
obj.m3()
```

```
m1 method
m2 method
m3 method
```

In [ ]:
```
#hierarichial inheriterance
class vehical:
  def disp1(self):
    print("vehical information")
class car(vehical):
  def disp2(self):
    print("car information")
class areoplane(vehical):
  def disp3(self):
    print("Areoplane information")
obj1=vehical()
obj1.disp1()
obj1.disp2()
obj1.disp3()
```

```
vehical information
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-520-aae7dd835ad4> in <cell line: 13>()
     11 obj1=vehical()
     12 obj1.disp1()
---> 13 obj1.disp2()
     14 obj1.disp3()

AttributeError: 'vehical' object has no attribute 'disp2'
```

In [ ]:
```
#hierarichial inheriterance
class vehical:
  def disp1(self):
    print("vehical information")
class car(vehical):
  def disp2(self):
    print("car information")
class areoplane(vehical):
  def disp3(self):
    print("Areoplane information")
obj1=vehical()
obj1.disp1()
```

```
vehical information
```

In [ ]:
```
obj2=car()
obj2.disp2()
```

```
car information
```

In [ ]:
```
obj2.disp1()
```

```
vehical information
```

In [ ]:
```
obj2.disp3()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-525-90d9e1734828> in <cell line: 1>()
----> 1 obj2.disp3()

AttributeError: 'car' object has no attribute 'disp3'
```

In [ ]:
```
obj3=areoplane()
obj3.disp1()
```

```
vehical information
```

In [ ]:
```
obj3.disp2()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-527-aba113b6ce38> in <cell line: 1>()
----> 1 obj3.disp2()

AttributeError: 'areoplane' object has no attribute 'disp2'
```

In [ ]:
```
obj3.disp3()
```

```
Areoplane information
```

In [ ]:
```
class vehical: #case 1
  def disp1(self):
    print("vehical information")
class car(vehical):
  def disp2(self):
    print("car information")
class areoplane(vehical):
  def disp3(self):
    print("Areoplane information")
obj=vehical()
obj.disp1()
"""obj.disp2() Will not be able to inherete
obj.disp3()"""
```

```
vehical information
```

Out[ ]:
```
'obj.disp2() Will not be able to inherete\nobj.disp3()'
```

In [ ]:
```
class vehical: #case 2
  def disp1(self):
    print("vehical information")
class car(vehical):
  def disp2(self):
    print("car information")
class areoplane(vehical):
  def disp3(self):
    print("Areoplane information")
obj1=car()
obj1.disp1()
obj1.disp2()
#obj1.disp3() #will not be able to inherte
```

```
vehical information
car information
```

In [ ]:
```
class vehical: #case 2
  def disp1(self):
    print("vehical information")
```

```
class car(vehical):
  def disp2(self):
    print("car information")
class areoplane(vehical):
  def disp3(self):
    print("Areoplane information")
obj2=areoplane()
obj2.disp1()
#obj2.disp2() #will bot be able to inherate
obj2.disp3()
```

```
vehical information
Areoplane information
```

In [ ]:
```
#multiple Inheriterance
class parent1():
  def m1(self):
    print("Parent1 information")
class parent2():
  def m2(self):
    print("parent2 information")
class child(parent1,parent2):
  def m3(self):
    print("child Information")
obj=child()
obj.m1()
obj.m2()
obj.m3()
```

```
Parent1 information
parent2 information
child Information
```

In [ ]:
```
class School:
    def func1(self):
        print("This function is in school.")
class Student1(School):
    def func2(self):
        print("This function is in student 1. ")
class Student2(School):
    def func3(self):
        print("This function is in student 2.")
class Student3(Student2, School):
    def func4(self):
        print("This function is in student 3.")
object = Student3()
object.func1()
object.func3()
```

```
This function is in school.
This function is in student 2.
```

In [ ]:
```
class person():
  def __init__(self,fname,lname):
    self.fname=fname
    self.lname=lname
class emp(person):
  def __init__(self,fname,lname,eid):
    self.fname=fname
    self.lname=lname
    self.eid=eid
  def disp(self):
    print("employee id number is:",self.eid)
    print("employee first name is:",self.fname)
    print("employee last name is:",self.lname)
```

```
obj=emp("sanad","singh",2457)
obj.disp()
```

```
employee id number is: 2457
employee first name is: sanad
employee last name is: singh
```

In [ ]:
```
class person(): #use of super function
  def __init__(self,fname,lname):
    self.fname=fname
    self.lname=lname
class emp(person):
  def __init__(self,fname,lname,eid):
    #self.fname=fname
    #self.lname=lname
    super().__init__(fname,lname)
    self.eid=eid
  def disp(self):
    print("employee id number is:",self.eid)
    print("employee first name is:",self.fname)
    print("employee last name is:",self.lname)
obj=emp("sanad","singh",2457)
obj.disp()
```

```
employee id number is: 2457
employee first name is: sanad
employee last name is: singh
```

In [ ]:
```
class person():
  def __init__(self,fname,lname):
    self.fname=fname
    self.lname=lname
class emp(person):
  def __init__(self,fname,lname,eid):
    #self.fname=fname
    #self.lname=lname
    #super().__init__(fname,lname)
    person.__init__(self,fname,lname) #use of parent class.
    self.eid=eid
  def disp(self):
    print("employee id number is:",self.eid)
    print("employee first name is:",self.fname)
    print("employee last name is:",self.lname)
obj=emp("sanad","singh",2457)
obj.disp()
```

```
employee id number is: 2457
employee first name is: sanad
employee last name is: singh
```

In [ ]:
```
class Mammal():
  def __init__(self, mName):
    print(mName, 'is a warm-blooded animal.')
class Dog(Mammal):
  def __init__(self):
    print('Dog has four legs.')
    super().__init__('Dog') #use of super keyword in single level inheritance.
d1 = Dog()
```

```
Dog has four legs.
Dog is a warm-blooded animal.
```

Abstract Class

- An abstract class can be considered a blueprint for other classes.
- It allows you to create a set of methods that must be created within any child classes built from the abstract class.
- A class that contains one or more abstract methods is called an abstract class.
- An abstract method is a method that has a declaration but does not have an implementation.
- We use an abstract class while we are designing large functional units or when we want to provide a common interface for different implementations of a component.

Abstract Base Classes in Python

- By defining an abstract base class, you can define a common Application Program Interface(API) for a set of subclasses.
- This capability is especially useful in situations where a third party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code base where keeping all classes in your mind is difficult or not possible.

In [ ]:
```
import abc

class dataloves :

    @abc.abstractmethod
    def student_details(self):
        pass

    @abc.abstractmethod
    def student_assignment(self):
        pass


    @abc.abstractmethod
    def student_marks(self):
        pass
```

In [ ]:
```
class data_science(dataloves):

    def student_details(self):
        return "it will try to return a details of data science masters "

    def student_assignment(self):
        return "it will return a details of student assignemnt for data science mas
```

In [ ]:
```
class web_dev(dataloves):
     def student_details(self):
            return "this will retrun a detils of web dev "


     def student_marks(self):
             return "this will return a makrs of web dev class"
```

In [ ]:
```
ds = data_science()
ds.student_details()
```

Out[ ]:
```
'it will try to return a details of data science masters '
```

In [ ]:
```
wb = web_dev()
wb.student_details()
```

Out[ ]:
```
'this will retrun a detils of web dev '
```

In [ ]:
```
from abc import abstractmethod #method 1 abc=package or module that is predefined A
from abc import ABC,abstractmethod #method 2
class person(ABC):
  def m1(self):
    pass
class tcs(person):
  def eat(self):
    print("tcs  eat 5 employe")
class ibm(person):
  def eat(self):
    print("ibm eat 50 emolye")
obj=ibm()
obj.eat()
obj=tcs()
obj.eat()
```

```
ibm eat 50 emolye
tcs  eat 5 employe
```

In [ ]:
```
from abc import ABC,abstractmethod
class A(ABC):
  @abstractmethod
  def disp1(self):
    pass
  @abstractmethod
  def disp2(self):
    pass
class himanshu(A):
  def disp1(self):
    print("disp1 implimentation")
  def disp2(self):
     print("kokok")
obj=himanshu()
obj.disp2()
obj.disp1()
```

```
kokok
disp1 implimentation
```

In [ ]:
```
class my:
  def __disp1(self):
    print("my self veer")
  def disp2(disp1):
    print("follow me please")
obj=my()
obj.disp2()
#obj.disp1() not able to fetch disp1 because its secure
```

```
follow me please
```

In [ ]:
```
from abc import ABC,abstractmethod
class BMW(ABC):
  @abstractmethod
```

```
  def disp(self):
    pass
class X3(BMW):
  def disp(self):
    print("very good design")
obj1=BMW()
obj1.disp()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-571-8fd965a8d78c> in <cell line: 9>()
      7   def disp(self):
      8     print("very good design")
----> 9 obj1=BMW()
     10 obj1.disp()

TypeError: Can't instantiate abstract class BMW with abstract method disp
```

In [ ]:
```
obj2=X3()
obj2.disp()
```

```
very good design
```

Encapsulation in Python

- Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit.
- This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data.
- To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.
- A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.
- The goal of information hiding is to ensure that an object's state is always valid by controlling access to attributes that are hidden from the outside world.

In [ ]:
```
class test :
    def __init__(self , a,b ) :
        self.a = a
        self.b = b
```

In [ ]:
```
t = test(45,56)
```

In [ ]:
```
print(t.a)
t.b
```

Out[ ]:
```
45
56
```

In [ ]:
```
t.a = 3243
print(t.a)
```

```
3243
```

In [ ]:
```
class bank_acount:
```

```
    def __init__(self , balance ):
        self.__balance = balance

    def deposit(self , amount ) :
        self.__balance = self.__balance + amount

    def withdraw(self , amount) :
        if self.__balance >= amount :
            self.__balance = self.__balance -amount
            return True
        else :
            return False

    def get_balance(self) :
        return self.__balance
obj_bank_account = bank_acount(1000)
obj_bank_account.get_balance()
```

Out[ ]: 1000

In [ ]:
```
obj_bank_account.deposit(6000)
```

In [ ]:
```
obj_bank_account.get_balance()
```

Out[ ]: 7000

In [ ]:
```
obj_bank_account.withdraw(10000)
```

Out[ ]: False

In [ ]:
```
obj_bank_account.withdraw(2000)
```

Out[ ]: True

In [ ]:
```
obj_bank_account.get_balance()
```

Out[ ]: 5000

Polymorphism in Python

- The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types.
- The key difference is the data types and number of arguments used in function.

In [ ]:
```
def test(a,b) :
    return a+b
```

In [ ]:
```
test(4,5)
```

Out[ ]: 9

In [ ]:
```
test("sanad" , "kumar")
```

Out[ ]: 'sanadkumar'

In [ ]:
```
test([2,3,4,5,5] , [4,5,6,7])
```

Out[ ]:
```
[2, 3, 4, 5, 5, 4, 5, 6, 7]
```

In [ ]:
```
class data_science:
    def syllabus(self) :
        print("this is my method for data science syllbaus " )
```

In [ ]:
```
class web_dev :
    def syllabus(self) :
        print("this is my method for web dev " )
```

In [ ]:
```
def class_parcer(class_obj) :
    for i in class_obj :
        i.syllabus()
```

In [ ]:
```
obj_data_science= data_science()
```

In [ ]:
```
obj_web_dev = web_dev()
```

In [ ]:
```
class_ojb = [obj_data_science , obj_web_dev]
```

In [ ]:
```
class_parcer(class_ojb)
```

```
this is my method for data science syllbaus
this is my method for web dev
```

In [ ]: