1. What is Python and why is it called an interpreted language?

Ans: Python is a high-level, general-purpose programming language known for its readable syntax and versatility in applications like web development, data science, automation, and software development.It was created by Guido van Rossum and released in 1991, and is widely used due to its simplicity and extensive standard libraries.

- Python is called an interpreted language because its code is executed line-by-line by the Python interpreter at runtime, instead of being compiled entirely into machine code before execution as is the case with compiled languages like C or C++.
- Python performs parsing and compiles the source code to an intermediate bytecode, which is then executed by the Python Virtual Machine (PVM).
- This approach allows for dynamic execution and easier debugging, as errors are reported line by line and developers can immediately see the result of their code without compiling the whole program.
- The line-by-line execution also makes Python more portable and flexible for scripting and rapid testing.

---

2. What are the key features of Python that make it popular for beginners and professionals?

Ans:Python is popular among both beginners and professionals due to its simple syntax, versatility, and extensive library support, making coding more accessible, efficient, and powerful.

Key Features That Make Python Popular

1. Simple and Easy to Learn : Python uses a clear, English-like syntax, reducing code complexity and making it highly readable even for those new to programming.
2. Free and Open Source : Python is freely available to download, use, and modify, encouraging wide community adoption and collaborative development.
3. Cross-Platform Compatibility: Python code runs on multiple operating systems, including Windows, Linux, and macOS, with little or no modification, making it portable across different computing environments.
4. Interpreted and Dynamically Typed: As an interpreted language, Python executes code line-by-line, simplifying debugging and encouraging experimentation.
5. Object-Oriented and Multi-Paradigm Support: Python supports object-oriented, procedural, and functional programming styles, letting developers choose the best paradigm for their task.
6. Large Standard Library: Python includes a vast standard library for tasks like file I/O, math, web development, and more, reducing the need to write code from scratch or rely on third-party modules.
7. High-Level Language: Python handles many low-level details (e.g., memory management) automatically, letting programmers focus on solving problems rather than managing technical complexities.

8. GUI and Integration Support: Python supports rapid development of graphical user interfaces (GUI) through libraries like Tkinter and PyQT.

---

3. What is the difference between Python 2 and Python 3?

Ans: The main difference between Python 2 and Python 3 lies in their syntax, features, and long-term support, with Python 3 being the modern, actively maintained, and recommended version.

1. Syntax Changes

- Print Statement: Python 2 uses print "Hello" (statement), while Python 3 uses print("Hello") (function), requiring parentheses.
- Integer Division: Dividing two integers in Python 2 (7/2) yields 3, but in Python 3, this yields 3.5 (floating-point result); floor division in Python 3 uses //.
- Unicode Handling: Strings are ASCII by default in Python 2 and Unicode by default in Python 3, making internationalization easier in Python 3.

2. Libraries and Compatibility

- Library Support: Many new libraries work only with Python 3. Python 2 libraries are not forward compatible with Python 3.
- Backward Compatibility: Python 3 is not backward compatible—Python 2 code usually must be adapted to run under Python 3.

3. Iteration

- xrange vs range: Python 2 has xrange() for efficient looping; Python 3 replaces it with only range(), which returns a generator-like object.

4. Exception Handling

- Syntax: Python 2 uses except Exception, e, while Python 3 uses except Exception as e, standardizing exception syntax.

5. Usage & Support

- End of Life: Python 2 reached its end of life in 2020 and is no longer maintained, while Python 3 is actively developed and recommended for all new projects.

---

4. What are Python’s applications in real-world projects?

Ans: Python is widely used for web development, data science, automation, AI, and many other fields, making it essential for both technical and business real-world projects.

Python Applications in Real-World Projects

1. Web Development

- Python powers dynamic websites and web applications with frameworks like Django, Flask, and Pyramid, used by companies such as Instagram and Pinterest.

- It is suitable for both simple microservices and large-scale enterprise solutions thanks to its robust libraries and scalability.

2. Data Science, Machine Learning, and AI

- Python is the standard language for data analysis, machine learning, and artificial intelligence, employing libraries like Pandas, NumPy, TensorFlow, and PyTorch.
- Applications range from financial forecasting and fraud detection (e.g., JP Morgan and Citigroup) to personalized Spotify playlists and autonomous vehicles.

3. Automation and Scripting

- Python excels in automating repetitive tasks, system administration, and DevOps, using libraries like Selenium and PyAutoGUI.
- Automation spans industries from personal productivity (like spreadsheet management) to industrial robotics and workflow automation.

4. Game Development

- Python is used for developing games, with engines like Pygame and integration in larger gaming platforms.

5. Business and Enterprise Solutions

- Python supports ERP, CRM, and business applications due to its security, scalability, and vast libraries.
- It enables cloud computing and API integrations, powering backend services and microservices on platforms like AWS and Google Cloud.

6. CAD and Scientific Applications

- Python is used in computer-aided design for creating 2D and 3D models with tools like Blender, FreeCAD, and Open Cascade, and for scientific computing in biology, engineering, and astronomy.

---

5. What is PEP 8 and why is it important in Python programming?

Ans: PEP 8 is the official style guide for Python code, providing conventions on writing clean, readable, and consistent Python programs. It was authored in 2001 by Guido van Rossum, Barry Warsaw, and Alyssa Coghlan to enhance code readability and maintainability across the Python community.

- PEP stands for Python Enhancement Proposal, and PEP 8 specifically focuses on coding style guidelines such as indentation, naming conventions, line length, whitespace usage, and comment formatting.
- It prescribes rules like using 4 spaces for indentation, limiting lines to 79 characters, using snake_case for variable and function names, and CamelCase for class names.
- PEP 8 also sets recommendations on breaking long lines, spacing around operators, and how to write meaningful comments and docstrings.

Why is PEP 8 Important?

- Readability: Python code is read far more often than it is written. PEP 8 helps programmers write code that is easy to read and understand, reducing cognitive load.
- Consistency: By adopting uniform coding styles, teams and open-source projects can ensure that code looks familiar and coherent, facilitating collaboration and smoother code reviews.
- Maintainability: Well-styled code is easier to debug, refactor, and extend over time.
- Professionalism: Following PEP 8 signals attention to detail and adherence to community standards, reflecting well on programmers in professional environments.

---

6. Who developed Python and in which year was it released?

Ans: Python was developed by Guido van Rossum, a Dutch programmer, and it was first released in 1991. He started working on Python in December 1989 while at Centrum Wiskunde & Informatica (CWI) in the Netherlands, initially as a hobby project during Christmas time. The name "Python" was inspired by the British comedy series Monty Python's Flying Circus, which he enjoyed.

---

7. What do you mean by “dynamically typed” in Python?

Ans: In Python, "dynamically typed" means that the type of a variable is determined at runtime rather than at the time the code is written or compiled. This means variables do not have fixed types; instead, the Python interpreter infers the type based on the value assigned to the variable when the program is running.

Explanation of Dynamic Typing in Python:

- You do not need to declare a variable’s type explicitly; the type is inferred automatically from the assigned value.
- A variable can be assigned different types of values during program execution. For example, a variable might initially hold an integer and later be reassigned to a string or a list.
- Types are checked at runtime, so if you perform operations on incompatible types, Python will raise an error while the program runs (not at compile time).
- This feature offers great flexibility and rapid development but requires careful attention to avoid runtime type errors.

---

8. What is the difference between a compiler and an interpreter, and which does Python use?

Ans: A compiler and an interpreter both translate high-level programming code into machine-readable code, but they do so in fundamentally different ways:

- A compiler translates the entire source code of a program into machine code before execution, while an interpreter translates and executes the source code line-by-line at runtime.

- Compiled programs run faster because the translation happens once, while interpreted programs run slower due to ongoing translation during execution.
- A compiler generates an independent executable file, but an interpreter does not produce a separate machine code file and requires the source code for execution.
- Compilers detect all errors together after compiling the whole program, whereas interpreters display errors one at a time as they encounter each line.
- Compilers generally take more time to analyze the entire program upfront, while interpreters process code more quickly but execute it slower.
- Compilers use more memory for storing the compiled code, whereas interpreters use less memory as they translate code on the fly.
- Compiled languages include C, C++, Java; interpreted languages include Python, Ruby, and Perl.
- Compilers are better suited for production environments due to efficient execution, while interpreters are preferred in development environments for easier debugging and testing.

Python's Use

- Python primarily uses an interpreter. Its source code is first compiled into intermediate bytecode, then executed by the Python Virtual Machine (PVM) line-by-line, offering dynamic execution and easier debugging.
- This interpreter-based approach supports Python’s features like dynamic typing and interactive coding, but results in generally slower execution compared to fully compiled languages.

Print: The **print function in Python** is a built-in function used to display output to the console or screen.

Definition of Print in Python

The print() function prints the specified message or object(s) to the screen; it can handle strings, numbers, or any object, converting them to a string before displaying.

Example:

```
print("Good Morning")
Ans: Good Morning
```

Example:

```
print("hello good morning")

Name = str(input("Enter your name:"))

Age = int(input("enter your age:"))
```

```
print(Name,type(Name))

print(Age,type(Age))
```

```
Ans: hello good morning

Enter your name: Lohith

enter your age: 28

Lohith <class 'str'>

28 <class 'int'>
```

---

1) Write a program to read employee data

emp Id,emp Name,emp Salary.

expected output format:

Employee ID = 123

Employee Name = Priyanka

Employee Salary = 45000Rs

<u>**Input**</u>:

```
Id = int(input("Enter Employee ID"))

Name = str(input("Enter Employee Name"))

Salary = float(input("Enter Employee Salary"))

print(Id,type(Id))

print(Name,type(Name))

print(Salary,type(Salary))
```

<u>**Output**</u>:

```
Enter Employee ID 123

Enter Employee Name Priyanka

Enter Employee Salary 45000

123 <class 'int'>

Lohith <class 'str'>

45000.0 <class 'float'>
```

Method 1

Input: Id = int(input("Enter Employee ID"))

Name = str(input("Enter Employee Name"))

Salary = float(input("Enter Employee Salary"))

print(Id,type(Id))

print(Name,type(Name))

print(Salary,type(Salary))

print("Method 1")

print("Employee ID = ",id,"\nEmployee Name = ",Name,"\nEmployee Salary = ",Salary,"Rs")

Output: Enter Employee ID 123

Enter Employee Name Lohith

Enter Employee Salary 5000

123 <class 'int'>

Lohith <class 'str'>

5000.0 <class 'float'>

Method 1

Employee ID = <built-in function id>

Employee Name = Lohith

Employee Salary = 5000.0 Rs

. Format method

Input: Id = int(input("Enter Employee ID"))

Name = str(input("Enter Employee Name"))

Salary = float(input("Enter Employee Salary"))

print(Id,type(Id))

print(Name,type(Name))

print(Salary,type(Salary))

print(". Format methiod")

print("Emoployee id = {}\nEmployee name = {}\nemplyee salary = {}Rs\n".format(Id,Name,Salary))

Output: Enter Employee ID 123

```
Enter Employee Name Lohith
Enter Employee Salary 45000
123 <class 'int'>
Lohith <class 'str'>
45000.0 <class 'float'>
. Format methiod
Emoployee id = 123
Employee name = Lohith
emplyee salary = 45000.0Rs
```

F String format

Input:
```
Id = int(input("Enter Employee ID"))
Name = str(input("Enter Employee Name"))
Salary = float(input("Enter Employee Salary"))
print(Id,type(Id))
print(Name,type(Name))
print(Salary,type(Salary))
print("f string format")
print(f"Employee Id = {Id}\nEmployee name = {Name}\nEmployee Salary = {Salary}Rs")
```

Output:
```
Enter Employee ID 123
Enter Employee Name Lohith
Enter Employee Salary 45000
123 <class 'int'>
Lohith <class 'str'>
45000.0 <class 'float'>
f string format
Employee Id = 123
Employee name = Lohith
```

Employee Salary = 45000.0Rs

Coments

It is a part of code but it wont take any place in execution.

Types of comments:

1. Single line Comment: #

# My 1st Program

2. Multiline Comment: ''' ''' or """ """

""" Write a program to calculate student result"""

Keywords

They are special reserved words that have special meaning and cannot be used as identifiers.

(Variables, Function name etc...)

Exa: True, False, Try, For, Else ...etc

Datatypes:

it mention what types as data it was

Str, Int, Float, Bool, Complex

Sequence datatypes (Data Strucutre)

Str, List, Tuple, Set, Dict

Variables: Variables are the names which we use to store our values.

Rules to declare variable

Valid Variable declaration:

a=2

A=4

num = 20

Num1 = 31

Num2 = 90

NUM3 = 12

Stuid = 123

Stu_Id = 127

_stu_name = "Priya"

a,b,c,= 4,6,1

x=8; y=2; z=3;

Invalid variable declaration:

1num=5

Stu name="Karthik"

@num$=8

#Input/Output Function

Print()-Output function

Input()-Input function

Write a program to read student Id,name and 3 subject marks and display there details

Student ID

Student Name

Sub1 marks

Sub2 marks

Sub3 marks

Total marks

Percentage

<u>Input</u>

```
Student_id = input("Enter student ID:")

Student_name = input("Enter Student name:")

s1 = float(input("Enter marks s1:"))

s2 = float(input("Enter marks s2:"))

s3 = float(input("Enter marks s3:"))
```

```
Total_marks = s1+s2+s3

Percentage = (Total_marks /300)*100

print(f"Student ID:{Student_id}\nStudent name:{Student_name}\ns1:{s1}\ns2:{s2}\ns3:{s3}\nTotal marks:{Total_marks}\nPercentage:{Percentage}%")
```

Output

```
Enter student ID: 143

Enter Student name: Lohith

Enter marks s1: 43

Enter marks s2: 56

Enter marks s3: 81

Student ID:143

Student name:Lohith

s1:43.0

s2:56.0

s3:81.0

Total marks:180.0

Percentage:60.0%
```

Operators

it is a special symbol perform certain operation b/w operands

ex: a = 3

= Operator a,3 operands

z= x+y

=+ operator x,y,z operands

Types of operator

1) Arithmetic operators

+ - * / % // **

Example:

Input:

```
a = float(input("Enter A value:"))

b = float(input("Enter B value:"))

print(f"the sum of {a} and {b} is {a+b}\nthe diff of {a} and {b} is {a-b}\nthe diff of {a} and {b} is {a-b}\nthe mul of {a} and {b} is {a*b}\nthe div of {a} and {b} is {a/b}\nthe mod of {a} and {b} is {a%b}\nthe floor div of {a} and {b} is {a//b}\nthe exp of {a} and {b} is {a**b}")
```

Output:

```
Enter A value: 6

Enter B value: 4

the sum of 6.0 and 4.0 is 10.0

the diff of 6.0 and 4.0 is 2.0

the diff of 6.0 and 4.0 is 2.0

the mul of 6.0 and 4.0 is 24.0

the div of 6.0 and 4.0 is 1.5

the mod of 6.0 and 4.0 is 2.0

the floor div of 6.0 and 4.0 is 1.0

the exp of 6.0 and 4.0 is 1296.0
```

2) comparison/relational operators

< > <= >= == !=

Example 1:

take a 2 inputs from user and perform all arithmetic operator and print all the output

Input:

```
a = float(input("Enter A value:"))

b = float(input("Enter B value:"))

print(f"{a+b},{a-b},{a*b},{a/b},{a%b}")
```

Output:

```
Enter A value: 6

Enter B value: 4

10.0,2.0,24.0,1.5,2.0
```

Example 2:

<u>Input:</u> <u>Output:</u>

x = 20 y = 40

print(x<=y)	True
print(x>=y)	False
print(x==y)	False
print(x!=y)	True

Example 3:

print(x<y)	True
Print(x>y)	False

and = all the conditions should pass

or = atleast one condition should pass

not = vise versa

A and B is value

A	**B**	**And**	**Or**	**Xor**	**Xnor**
Fail	Fail	F	F	T	F
Fail	Pass	F	T	F	T
Pass	Fail	F	T	F	T
Pass	Pass	T	T	T	F

A	**Not**
T	F
F	T

3) logical operators

And, or, not

Example 1:

<u>Input</u>

a = 7

b = 9

Print(a>10 and b<10)	False
Print(a<10 and b<10)	True
Print(a>10 and b<10)	True
Print(a>10 and b>10)	False

Not

Not(a>10)	True
Not(a<10)	False

Write a Python program that takes the length and width of a rectangle from the user and prints its area.

Input:

```
length = float(input("Enter the length of the rectangle:"))

width = float(input("Enter the width of the rectangle:"))

area = length*width

print(f"the area of the rectangle is: {area}")
```

Output:

```
Enter the length of the rectangle: 152

Enter the width of the rectangle: 90

the area of the rectangle is: 13680.0
```

Write a program that asks the user for the side of a square and prints its perimeter.

Input:

```
side = float(input("Enter the side lenth of square:"))

Perimeter = 4 * side

print(f"the perimeter of the square is:",Perimeter)
```

Output:

```
Enter the side lenth of square: 160

the perimeter of the square is: 640.0
```

Take the base and height of a triangle as input and print its area.

Input:

```
base = float(input("Enter the base value:"))

height = float(input("Enter the height value:"))

area = 1/2*base*height

print(f"the triangle of the area is:",area)
```

Output:

```
Enter the base value: 60

Enter the height value: 30

the triangle of the area is: 900.0
```

Write a program that asks the user for the radius of a circle and prints its circumference. (Use 3.14 for π)

Input:

```
radius = float(input("enter the radius value of circle:"))

pi = 3.14

circumference = 2*pi*radius

print(f"The circumference of the circle is:",circumference)
```

Output:

```
enter the radius value of circle: 12

The circumference of the circle is: 75.36
```

Take Principal (P), Rate (R), and Time (T) as input from the user and print the Simple Interest.

Input:

```
p = float(input("Enter the principal value:"))

R = float(input("Enter the rate value:"))
```

```
t = int(input("Enter the time:"))

Simple_interest = (p*R*t)/100

print(f"The Simple interest is:",Simple_interest)
```

Output:

```
Enter the principal value: 350000

Enter the rate value: 3

Enter the time: 4

The Simple interest is: 42000.0
```

---

4) Assignment operators

= += -= *= /= //= **=

1. User entry :A=5
   Input: A

   Output:5
2. User entry : A += 4
   Input : A
   Output: 9
3. User entry :A -= 3
   Input: A
   Output: 6
4. User entry :A *= 5
   Input: A
   Output: 30
5. User entry : A /= 4
   Input: A
   Output: 7.5
6. User entry :A **= 5
   Input: A
   Output: 7.525434581650003e+21
7. User entry : A //= 5
   Input: A
   Output: 1.5050869163300007e+21

---

5) Identity operators

Is, is not

Example:

A=5 , B=6

Input: a is b ,Output: False

Input: a is not b, output : True

---

6) Membership operators

In , not in

i am the member of your family - false

i am not a member of your family - true

you are a member of your family - true

you are not a member of your family - false

Example:

Num={1,8,3,9,0}

Input: 1 in num	Output: true
Input: 2 in num	Output: False
Input: 8 not in num	output: False
Input: 3 in num	Output: true

---

conditional statments

It is allow us to make decisions in code. they check coditions (expression that result in True or False) and execute different blocks of code accordingly

types of conditional statments in python

1) if statment - executes a block only if the condition is true.

2) if..else statment - provides two paths : ine if codition is True another if False.

3) if..elif..else ladder - Multiple conditions checked one by one.

4) nested if - using one if inside another.

1. if statment - executes a block only if the condition is true

Systax: if(condition):

```
                statements
```

eligibility checking for election voting in india

Input:

```
age = int(input("Enter your name:"))
Country = input("Enter your country:")
if(age>=18 and Country =="india"):
   print("Eligible for voting")
else:
     ("not eligible for voting")
```

Output:

```
Enter your name: 20
Enter your country: india
Eligible for voting
```

2. if..else statment - provides two paths : ine if codition is True another if False

Syntax: If(condition):

```
                Statements
        Else:
            Statements
```

Example:

Input:

```
age = int(input("Enter your name:"))
Country = input("Enter your country:")
if(age>=18 and Country =="india"):
   print("Eligible for voting")
else:
    print("not eligible for voting")
```

Output:

```
Enter your name: 28
Enter your country: Germany
not eligible for voting
```

3. if..elif..else ladder - Multiple conditions checked one by one

Syntax:

```
if(condition1):
        statement of condtion1
else:
        if(condition2):
                statements of condition2
        else:
                if(condition3):
                        statements of condition3
                else:
                        else block statements
```

Syntax:

```
if(condition1):
        statement of condtion1
elif(condition2):
        statements of condition2
elif(condition3):
        statements of condition3
else:
        else block statements
```

---

take percentage from user and print their result

85-100 => distinction

60-84 => first class

50-59 => second class

35-49 => pass

0-34 => fail

Input:

```
per = float(input("Enter your percentage:"))
if(per>=85 and per<=100):
   print("distinction")
elif(per>=60 and per<=84):
   print("first class")
elif(per>=50 and per<=59):
   print("second class")
elif(per>=35 and per<=49):
   print("pass")
elif(per>=0 and per<=34):
   print("fail")
else:
   print("invalid input")
```

Output:

```
Enter your percentage: 73
first class
```

'''Write a program to check the given number is even or odd'''

Input:

```
n = int(input("Enter a value:"))
if(n%2==0):
   print("even")
else:
   print("odd")
```

Output:

```
Enter a value: -6
Even
```

'''write a program to check the given input is positive number, negative number or zero'''

Input:

```
n = int(input("Enter a number:"))
if(n<0):
   print("-ve number")
elif(n>0):
   print("+ve number")
else:
   print("Zero")
```

Output:

```
Enter a number: 5
+ve number
```

```
purchase_amount = float(input("Enter purchase amount:"))
if(purchase_amount <=100):
   discount = 0
elif(purchase_amount >= 100 and purchase_amount<=500):
   discount = purchase_amount* 0.10
else:
   purchase_amount >= 500
   discount = purchase_amount* 0.20
print("Discount:", discount)
print("Final_amount:", purchase_amount-discount)
```

Output:

```
Enter purchase amount: 560

Discount: 112.0

Final_amount: 448.0
```

```
Traffic_light = input("Enter a color:")
if(Traffic_light == "red"):
   print("Stop")
elif(Traffic_light == "Yellow"):
   print("Ready to move")
elif(Traffic_light == "Green"):
   print("Go")
else:
   print("Invalid color")
```

<u>Output:</u>

```
Enter a color: Yellow

Ready to move
```

---

6) Bitwise operator

- to do bitwise operation
- 2^n

  & | ~ >> <<

- by using these operators for bitwise operartors

1. And & operation

   Input:15&3

   Output:3

2. Or | operation
   Input:15|3
   Output:15
3. >> operation (rightshift)

Input: 13>>2
Output: 3

4. << operation (leftshift)
   Input: 13<<2
   Output: 52

---

nested if operators

syntax:

```
if(condition1): #outer if
   if(condition2): #inner if
     statement of condition2
   else: #inner if else
     statement of inner if else statements
else: #outer if else
  statements of outer if else
```

write a program to check a given input is positive number,negetive number or zero

Input:

```
n = int(input("Enter a value:"))
if(n>0):
    if(n==0):
     print("zero")
    else:
       print("Positive")
else:
    print("Negetive number")
```

Output:

```
Enter a value: -3
Negetive number
```

Loops

loops in python are used to execute a block of code repeatedly until a certain condition is met.

Python mainly have 2 loops:

1). for loop

2). while loop

3). nested loop

  * for loop inside for loop

  * for loop inside while loop

  * while loop inside while loop

  * while loop inside for loop

for loop : two types

1). for loop with sequence: string,list,tuple,set,dict

2). for loop with range :

1. example for for loop with sequence

   name = "Lohith"

   print(name)

```
for i in name: # i is ittirative variable
   print(i)
```

```
for i in enumerate(name): # enumerate it will give positioin and character of the variable
   print(i)
```

Example:

Input: fruit = ['apple','mango','grapes','pineapple']

```
        print(fruit)
```

Output: ['apple', 'mango', 'grapes', 'pineapple']

Input:for i in fruit:

```
        print(i)
```

output: apple

```
        mango
        grapes
        pineapple
```

2. for loop with range:
   initialization
   condition
   incrementation/decrementation

   range(start value,stop value, step size)

```
for i in range(1,11,1): # range(1,11,1)
   print(i)          # (i=1,i<11,i+1)
```

Example: for I in range (1,6,1):

```
                Print("lohith")
```

1)Lohith

2)Lohith

3)Lohith

4)Lohith

5)Lohith

```
for i in range(6): # default start value is 0
    print(i)       # default step size is 1
0
1
2
3
4
5
```

```
name = "Lohith"
for i in range(0,len(name),1):
    print(i,name[i])

0 L
1 o
2 h
3 i
4 t
5 h
```

---

write a program to print even numbers from 1 to 20

<u>input:</u>

```
for i in range(2,21,2):
    print(i)
```

<u>output:</u>

```
2
4
6
8
```

```
10

12

14

16

18

20
```

```
for i in range(1,11,1):
   if(i%2==0):
    print(i,"Even number")
   else:
      print(i,"Odd number")
```

Output:

```
1 Odd number

2 Even number

3 Odd number

4 Even number

5 Odd number

6 Even number

7 Odd number

8 Even number

9 Odd number

10 Even number
```