Unlocking Python: A Deep Dive into Essential Programming Concepts

CodroidHub summer training

Title:

1. Overview of Python
2. Difference between POP (Procedure-Oriented Programming) and OOP (Object-Oriented Programming)
3. Installation of PyScripter
4. Variables in Python
5. Single Line and Multi-Line Comments
6. Data Types in Python
7. Arithmetic Operators
8. Logical Operators
9. If-Else Condition
10. Match in Python

Submitted by: DRISHTI GUPTA

ROLL NO: (2322825)

BRANCH: AI&ML

SUBMITTED TO: Mr. DEVASHISH SIR

(FOUNDER, CodroidHub Private Limited)

Overview of Python :

- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. Python is an open-source community language, so numerous independent programmers are continually building libraries and functionality for it.

History of Python :

- Python was **created in 1980s by Guido van Rossum**. During his research at the **National Research Institute for Mathematics and Computer Science in the Netherlands,** he created Python – a super easy programming language in terms of reading and usage. The first ever version was released in the year 1991 which had only a few built-in data types and basic functionality.
- Later, when it gained popularity among scientists for numerical computations and data analysis, in 1994, Python 1.0 was released with extra features like map, lambda, and filter functions. After that adding new functionalities and releasing newer versions of Python came into fashion.
- *Python 1.5 released in 1997*
- *Python 2.0 released in 2000*
- *Python 3.0 in 2008 brought newer functionalities*

The latest version of Python, Python 3.12.4 – released on June 6, 2024.

Python Use Cases :

- Creating web applications on a server
- Building workflows that can be used in conjunction with software
- Connecting to database systems
- Reading and modifying files
- Performing complex mathematics
- Processing big data
- Fast prototyping
- Developing production-ready software

Conclusion :

- Python has a lot of reasons which make it a more popular and highly demanding programming language. High Community support and a large number of libraries and frameworks in Python make it the best choice for developers and beginners to choose it single handily. Python has use cases in web Development, Game Development, Automation, and technologies like AI, ML, and Data Analytics. Python is releasing its never version and adding new functions for the betterment of developers. But it has some limitations as well as discussed in the article like it is slow in execution, so Competitive Programmer prefer it less. But overall it is growing rapidly and has a very bright future ahead for this Programming language.

Example : PYTHON CODE :

python

```python
def factorial(n):
    # Base case: if n is 0, return 1
    if n == 0:
        return 1
    else:
        # Recursive case: n * factorial of (n-1)
        return n * factorial(n-1)

# Example usage
number = 5
# Print the factorial of the given number
print(f"The factorial of {number} is {factorial(number)}")
```

C++ CODE :

cpp

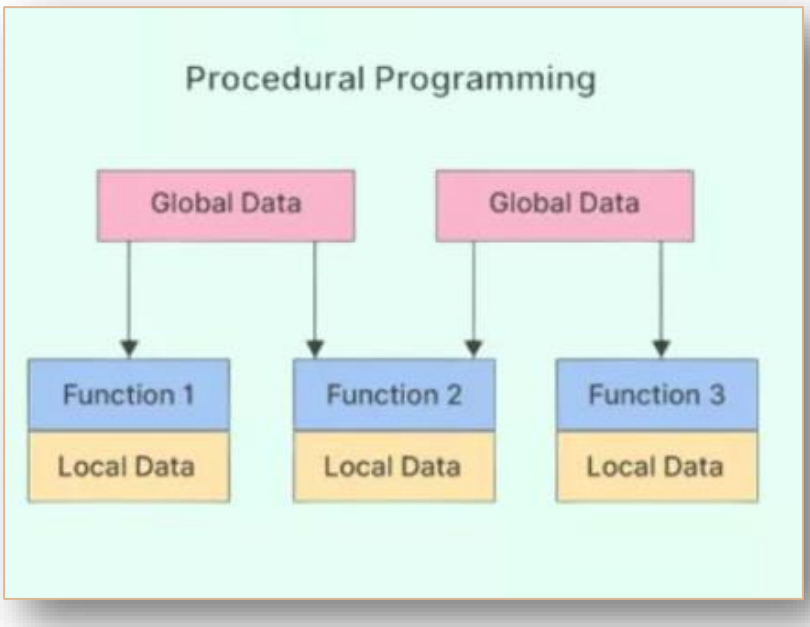
```cpp
#include <iostream>
using namespace std;

// Function to calculate factorial recursively
int factorial(int n) {
    // Base case: factorial of 0 is 1
    if (n == 0)
        return 1;
    else
        // Recursive case: n * factorial of (n-1)
        return n * factorial(n - 1);
}

int main() {
    // Example usage
    int number = 5;
    cout << "The factorial of " << number << " is " << factorial(number) << endl;

    return 0;
}
```

2. Difference between POP (Procedure-Oriented Programming) and OOP (Object-Oriented Programming) :

POPs – We can write the code sequentially.

CHARACTERISTICS :

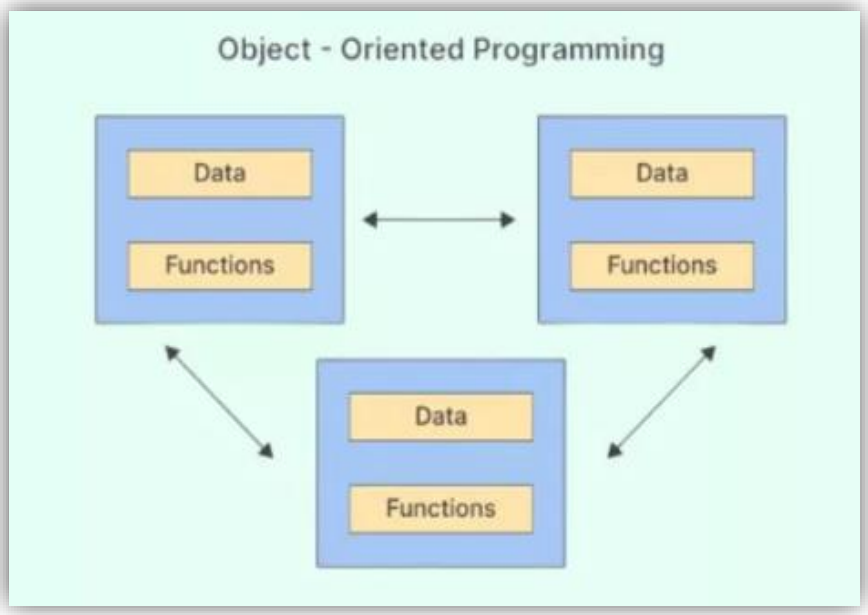
1. **Linear Structure**: Programs are typically written in a sequence from top to bottom.
2. **Modularization**: Code is divided into procedures or functions that perform specific tasks.
3. **Global Data**: Data is often shared globally, and procedures can access and modify global data.
4. **Function Focused**: The primary focus is on writing functions or procedures that operate on data.
5. **Reusability**: Procedures can be reused across different parts of the program.
6. **No Data Encapsulation**: Data is not bundled with the functions that operate on it.



OOPS – We can write code according to ourself but by following the rules and methods such as Abstraction, Inheritance and Polymorphism.

Characteristics :

1. **Encapsulation**: Data and the methods that operate on that data are bundled together into objects.
2. **Abstraction**: Hides complex implementation details and shows only the necessary features of an object.
3. **Inheritance**: Allows the creation of new classes based on existing classes, promoting code reuse.
4. **Polymorphism**: Allows methods to do different things based on the object it is acting upon.
5. **Modular and Reusable**: Objects and classes are modular and can be reused in different programs.
6. **Data Security**: Encapsulation helps in protecting data from unauthorized access and modification.



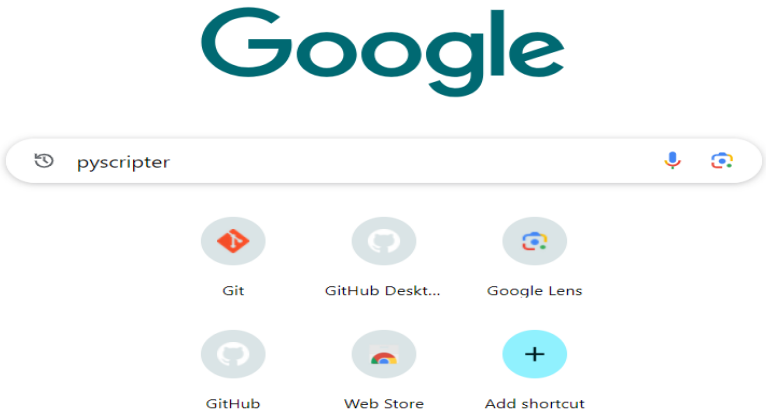
Key Differences :

Features	POPs	OOPs
Structure	Linear, function-based	Hierarchical, object-based
Modularity	Based on functions	Based on objects and classes
Data Handling	Global data shared across functions	Encapsulated within objects
Reusability	Function reuse	Class and object reuse
Data Security	Less secure, global data modification	More secure, encapsulation restricts access
Abstraction	No explicit abstraction mechanisms	Uses classes and objects to abstract real-world entities
Inheritance	No inheritance	Supports inheritance
Polymorphism	Not supported	Supported through method overloading and overriding
Focus	Functions and procedures	Objects and their interactions

3. Installation of PyScripter

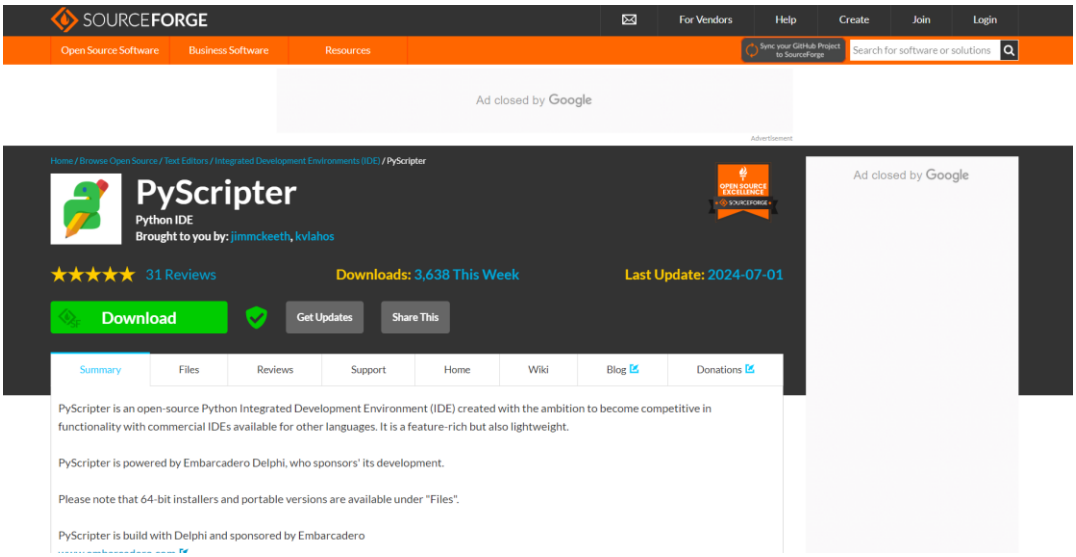
1. **Open Google Chrome**:

- Launch Google Chrome on your computer.



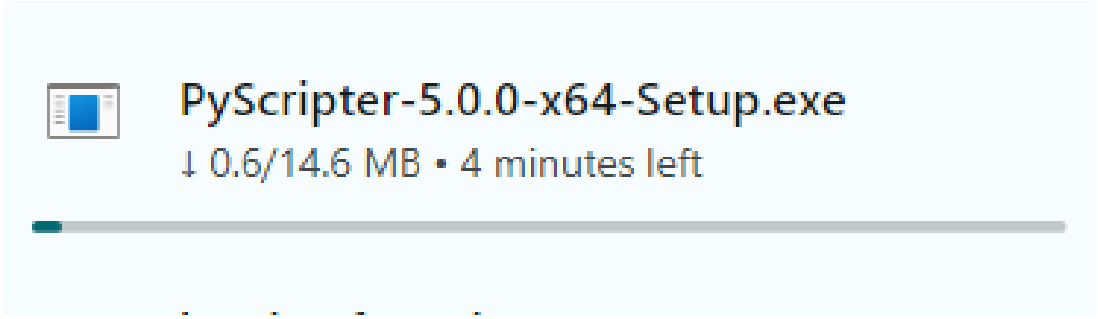
2. **Visit the PyScripter Website**:

- Go to the official PyScripter website https://sourceforge.net/projects/pyscripter/.



3. **Download the Installer**:

- On the PyScripter Source Forge page, locate the latest stable release.
- Click on the download link associated with the installer package (usually a `.zip` file).

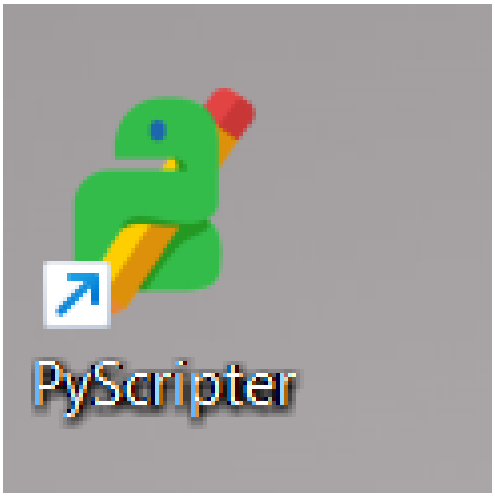


4. **Run the Installer**:

- Once the download is complete, locate the downloaded installer file (`.zip`).
- Extract the contents of the zip file to a folder on your computer.

5. **Launch PyScripter**:

- Navigate to the folder where you extracted the files.
- Find the executable file (usually named `PyScripter.exe` on Windows) and double-click to launch PyScripter.



4. Variables in Python

A variable is used to store data that can be referenced and manipulated in your code. Variables can hold different types of data, such as numbers, strings, lists, or even complex objects. Here's a basic overview of variables in Python:

```
#name is a variable
name="Drishti Gupta"
#age is a variable
age=20
print("my name is :",name)
print("my age is :",age)
```

Output :

```
*** Remote Interpreter Reinitialized ***
my name is : Drishti Gupta
my age is : 20
>>>
```

Variable Naming Rules :-

- Variable names must start with a letter (a-z, A-Z) or an underscore (_).
- The rest of the name can contain letters, numbers, or underscores.
- Variable names are case-sensitive (age and Age are different variables).
- Avoid using Python reserved words (keywords) as variable names (e.g., if, else, for, while).

5. Single Line and Multi-Line Comments

Single-line comment:

- Starts with `#`.
- It extends from the `#` symbol to the end of the line.

Example :

```
# This is a single-line comment
print("Hello, World!")  # This is also a comment
```

Multi-line comment (docstring):

- Enclosed in triple quotes (`"""` or `'''`).
- Can span multiple lines and are often used for documentation purposes.

Example:

```
"""
This is a multi-line comment or docstring.
It can be used to describe functions, modules, or classes.
"""
```

---

6. Data Types in Python

Data types are classifications that dictate how the interpreter will treat and store different kinds of data. Here are the primary built-in data types in Python.

Example:-

```
a = 20 #int
b = 4.5 #float
c = 2 + 6j #complex number
name = "Drishti" #str
numbers = [1,2,3,4] #list
coordinates = (20.0,40.0) #tuple
D1 = {"name":"Drishti","age":20} #dict
print(type(a))
print(type(b))
print(type(c))
print(type(name))
print(type(numbers))
print(type(coordinates))
print(type(D1))
print(a,b,c,name,numbers,coordinates,D1)
```

OUTPUT:

```
>>>
*** Remote Interpreter Reinitialized ***
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
20 4.5 (2+6j) Drishti [1, 2, 3, 4] (20.0, 40.0) {'name': 'Drishti', 'age': 20}
>>>
```

---

7. Arithmetic Operators

Arithmetic operators in Python are used to perform basic mathematical operations. Here is a list of the arithmetic operators available in Python along with examples:

Example:-

```
a=15
b=20
sum=a+b
diff=a-b
mult=a*b
div=a/b
mod=a%b
square=a**b
print (sum)
print(diff)
print(mult)
print(div)
print (mod)
print (square)
```

Output :

```
*** Remote Interpreter Reinitialized ***
35
-5
300
0.75
15
33252567300796508789062 5
>>>
```

---

8. Logical Operators

---

logic operator are used to combine multiple condition together and evaluate them as a single Boolean expression. There are three types of logical operators in python: and, or, not.

Example:-

```python
#not operator
a=10
b=20
print("return :",not(a>b))# apposite value if condition is true then it will return false and vice-versa

#and operator
value1 = True
value2 = True
print("return :",value1 and value2)  #both value are true then it will return true

#or operator
value1 = True
value2 = True
print("return :",value1 or value2) #if any ome of the value is true then it will return true
```

Output :

```
return : True
return : True
return : True
PS E:\vs code>
```

---

9. If-Else Condition

The if, Elif, and else statements are used to perform conditional logic. Here's a basic example to illustrate how these statements are used:

Here's a breakdown of each component:

1. **if Statement:** The if statement evaluates a condition. If the condition is True, the block of code under the if statement is executed.

2. **Elif Statement:** The Elif (short for "else if") statement is used to check multiple expressions for True and execute a block of code as soon as one of the conditions is True. You can have multiple Elif statements.

3**. else Statement:** The else statement is optional and catches anything which isn't caught by the preceding conditions. If none of the previous conditions are True, the block of code under the else statement is executed.

Example:-

```
age = 24
if (age < 18):
    print("you are minor")
elif (age>= 18 and age <50):
    print("you are an adult")
else:
    print("you are an senior")
```

Output :

```
you are an adult
PS E:\vs code>
```

---

10. Match in Python

In Python 3.10 and later, the match statement (similar to switch-case statements in other languages) is introduced for pattern matching. It allows for more readable and concise handling of multiple conditions.

Example:-

```python
day = int(input("enter the number :"))
match day:
    case 1:
        print("the day in monday")
    case 2:
        print("the day is tuesday")
    case 3:
        print("the day is wednesday")
    case 4:
        print("the day is thursday")
    case 5:
        print("the day is friday")
    case 6:
        print("the day is saturday")
    case 7:
        print("the day is sunday")
    case _:
        print("holiday")
```

Output:-

```
enter the number :8
holiday
PS E:\vs code>
```