PROGRAMMING IN PYTHON (CSM210-2C)

B.SC.CS SEM 4

UNIT 1

PYTHON INTRODUCTION

- Python is a high-level, interpreted programming language known for its simplicity and readability.
- It is created by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages in the world.
- It is widely used in various fields, including web development, data science, artificial intelligence, scientific computing, and more.

KEY FEATURES OF PYTHON

- **Easy to Learn and Use**: Python's syntax is clear and concise, making it an excellent choice for beginners. Its readability allows developers to write and understand code more efficiently.
- **Interpreted Language**: Python is an interpreted language, meaning that code is executed line by line, which makes debugging easier and faster
- **Dynamically Typed**: In Python, you don't need to declare the type of a variable. The type is determined at runtime, which adds flexibility to the code.

KEY FEATURES OF PYTHON

- **Extensive Standard Library:** Python comes with a vast standard library that includes modules and packages for various tasks, such as file I/O, system calls, and web development.
- **Cross-Platform Compatibility**: Python is available on multiple platforms, including Windows, macOS, and Linux. This allows developers to write code that can run on different operating systems without modification.
- **Object-Oriented Programming (OOP):** Python supports OOP, which allows for the creation of reusable code and modular programs. It also supports other programming paradigms, such as procedural and functional programming.

KEY FEATURES OF PYTHON

- **Large Community and Ecosystem**: Python has a large and active community that contributes to its development and provides support through forums, tutorials, and documentation. Additionally, there are numerous third-party libraries and frameworks available, such as Django for web development and TensorFlow for machine learning.
- **Integration Capabilities:** Python can easily integrate with other languages and technologies, such as C, C++, Java, and .NET, making it a versatile choice for various projects.
- **Open Source**: Python is open-source software, meaning it is free to use and distribute. Its source code is available for modification and customization.
- **Support for GUI Programming:** Python provides libraries like Tkinter, PyQt, and Kivy for developing graphical user interfaces (GUIs).

There are several excellent editors and tools available for Python programming, each with its own unique features and benefits.

Integrated Development Environments (IDEs)

1. **PyCharm**: A powerful and feature-rich IDE specifically designed for Python development. It offers intelligent code completion, code inspections, on-the-fly error highlighting, and quick-fixes, along with automated code refactoring and rich navigation capabilities.
2. **Spyder**: An open-source IDE that is particularly popular among data scientists. It integrates well with scientific libraries like NumPy, SciPy, and Matplotlib.
3. **IDLE**: The default IDE that comes with Python. It's simple and lightweight, making it a good choice for beginners.
4. **Visual Studio Code**: A free, open-source code editor developed by Microsoft. It supports Python through extensions and offers features like debugging, syntax highlighting, and code completion.
5. **Jupyter Notebook**: An open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It's widely used in data science and machine learning.

Code Editors

- **Sublime Text**: A sophisticated text editor with a wide range of features.
- **Atom**: A hackable text editor developed by GitHub.
- **Vim**: A highly configurable text editor for efficient text editing.
- **Emacs**: An extensible, customizable text editor with a powerful Python mode.
- **Thonny**: An IDE for beginners with a simple and clean interface.

Python is a versatile language used in various fields and industries.

Fields and Industries

1. **Web Development**:
    1. **Backend Developer**: Building server-side logic, APIs, and databases using frameworks like Django and Flask.
    2. **Full Stack Developer**: Working on both frontend and backend development.
2. **Data Science and Analytics**:
    1. **Data Scientist**: Analyzing and interpreting complex data to help companies make decisions.
    2. **Data Analyst**: Collecting, processing, and performing statistical analyses on data.
    3. **Machine Learning Engineer**: Developing algorithms and models for predictive analytics.
3. **Artificial Intelligence and Machine Learning**:
    - **AI Researcher**: Conducting research to advance the field of artificial intelligence.
    - **Machine Learning Engineer**: Implementing machine learning models and algorithms.

Python is a versatile language used in various fields and industries.

4. Automation and Scripting:

- **Automation Engineer**: Writing scripts to automate repetitive tasks and processes.
- **DevOps Engineer**: Automating deployment, scaling, and management of applications.

5. Scientific Computing:

- **Research Scientist:** Using Python for simulations, data analysis, and visualization in scientific research.
- **Bioinformatics Specialist**: Analyzing biological data using Python.

6. Finance and Trading:

- Quantitative Analyst: Developing models for trading strategies and risk management.
- Financial Analyst: Analyzing financial data and creating reports.

Python is a versatile language used in various fields and industries.

7. Game Development:

- **Game Developer**: Creating games using libraries like Pygame.

8. Education:

- **Instructor**: Teaching Python programming in schools, colleges, and online platforms.
- **Curriculum Developer**: Creating educational content and resources for learning Python.

Job Titles

1. Software Engineer
2. Backend Developer
3. Full Stack Developer
4. Data Scientist
5. Data Analyst
6. Machine Learning Engineer
7. AI Researcher
8. Automation Engineer
9. DevOps Engineer
10. Research Scientist

Job Titles

1. Bioinformatics Specialist
2. Quantitative Analyst
3. Financial Analyst
4. Game Developer
5. Instructor
6. Curriculum Developer

HOW TO SETUP PYTHON

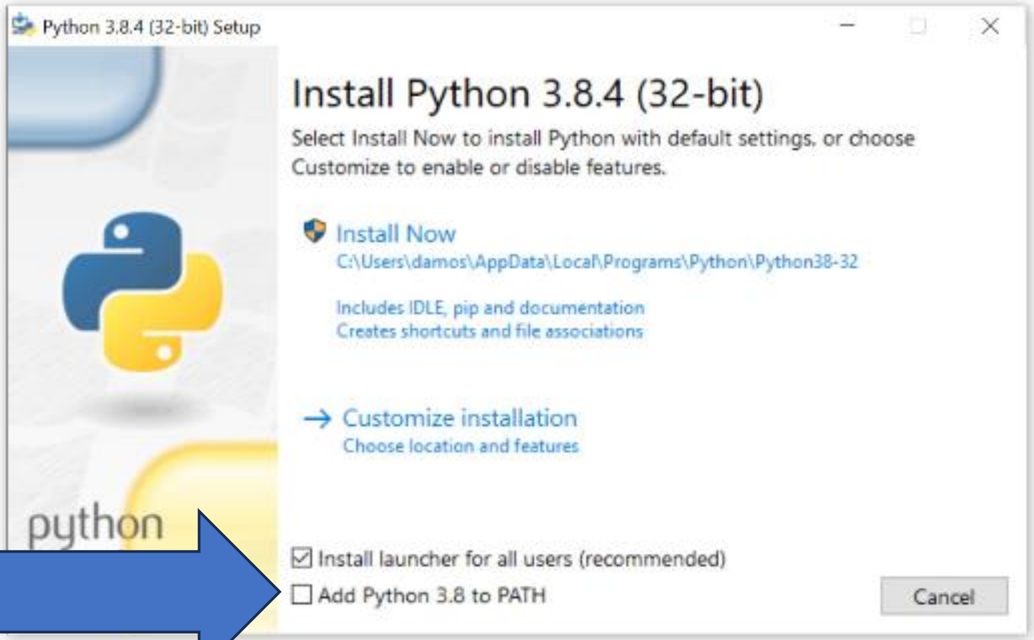
For Windows

Follow these steps to install Python 3 and open IDLE on Windows.

Step 1: Download the Python 3 Installer, Open a web browser and navigate to the following URL: https://www.python.org/downloads/windows/

Step 2: Run the Installer

Make sure you select the box that says Add Python 3.x to PATH.
If you install Python without selecting this box, then you can run the installer again and select it.



Python 3.8.4 (32-bit) Setup

Install Python 3.8.4 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\damos\AppData\Local\Programs\Python\Python38-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.8 to PATH

HOW TO SETUP PYTHON

Open IDLE

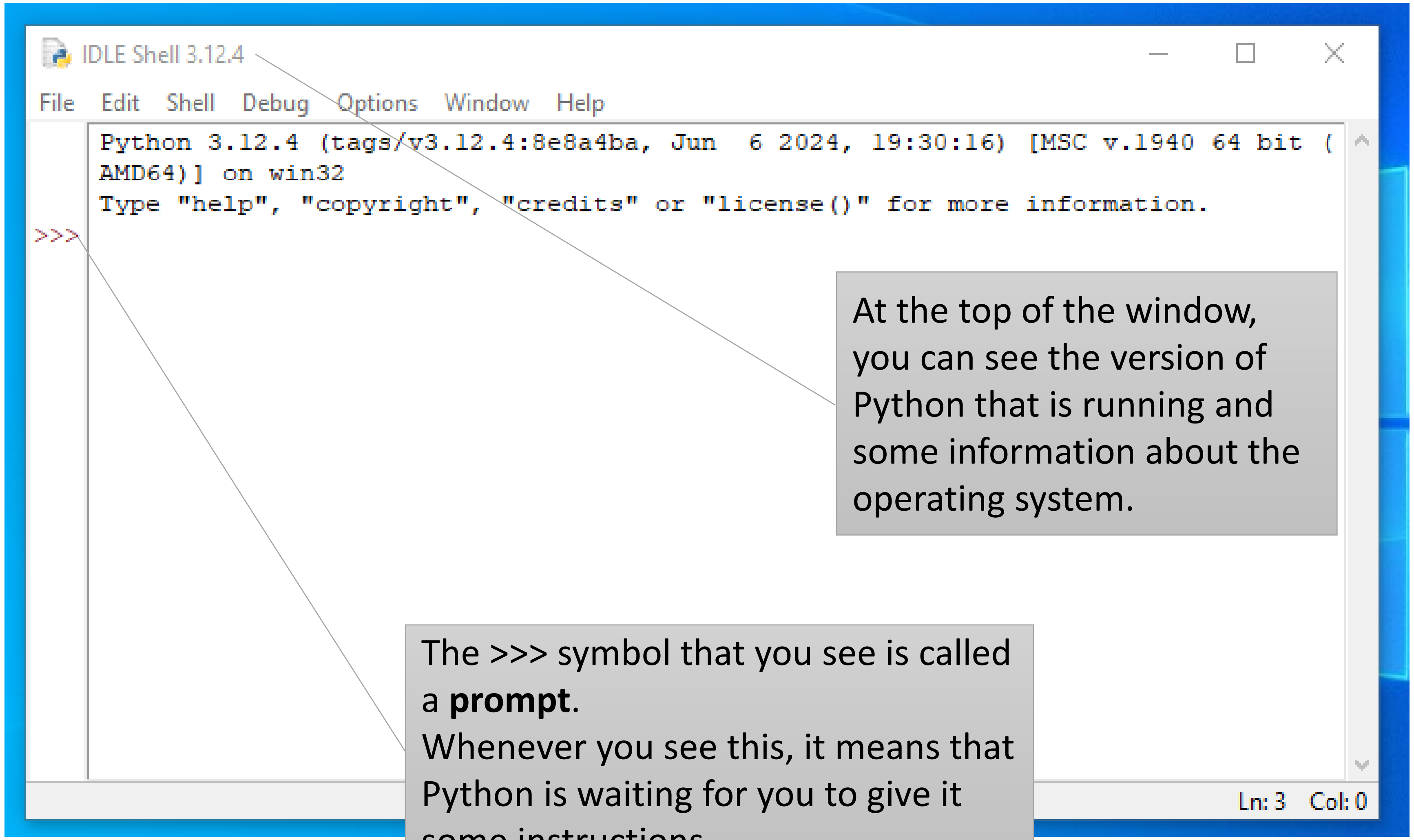
You can open IDLE in two steps:

1. Click the Start menu and locate the Python 3.X folder.
2. Open the folder and select IDLE (Python 3.X).

To get the path of python, open command prompt

```
C:\Users\student>where python

C:\Users\student\AppData\Local\Programs\Python\Python312\python.exe
C:\Users\student\AppData\Local\Microsoft\WindowsApps\python.exe
```



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

At the top of the window, you can see the version of Python that is running and some information about the operating system.

The >>> symbol that you see is called a **prompt**.
Whenever you see this, it means that Python is waiting for you to give it some instructions.

Write a program program

There are two main windows that you'll work with in IDLE:

the interactive window, which is the one that opens when you start IDLE,

and the editor window.

You can type code into both the interactive window and the editor window.

The difference between the two windows is in how they execute code.

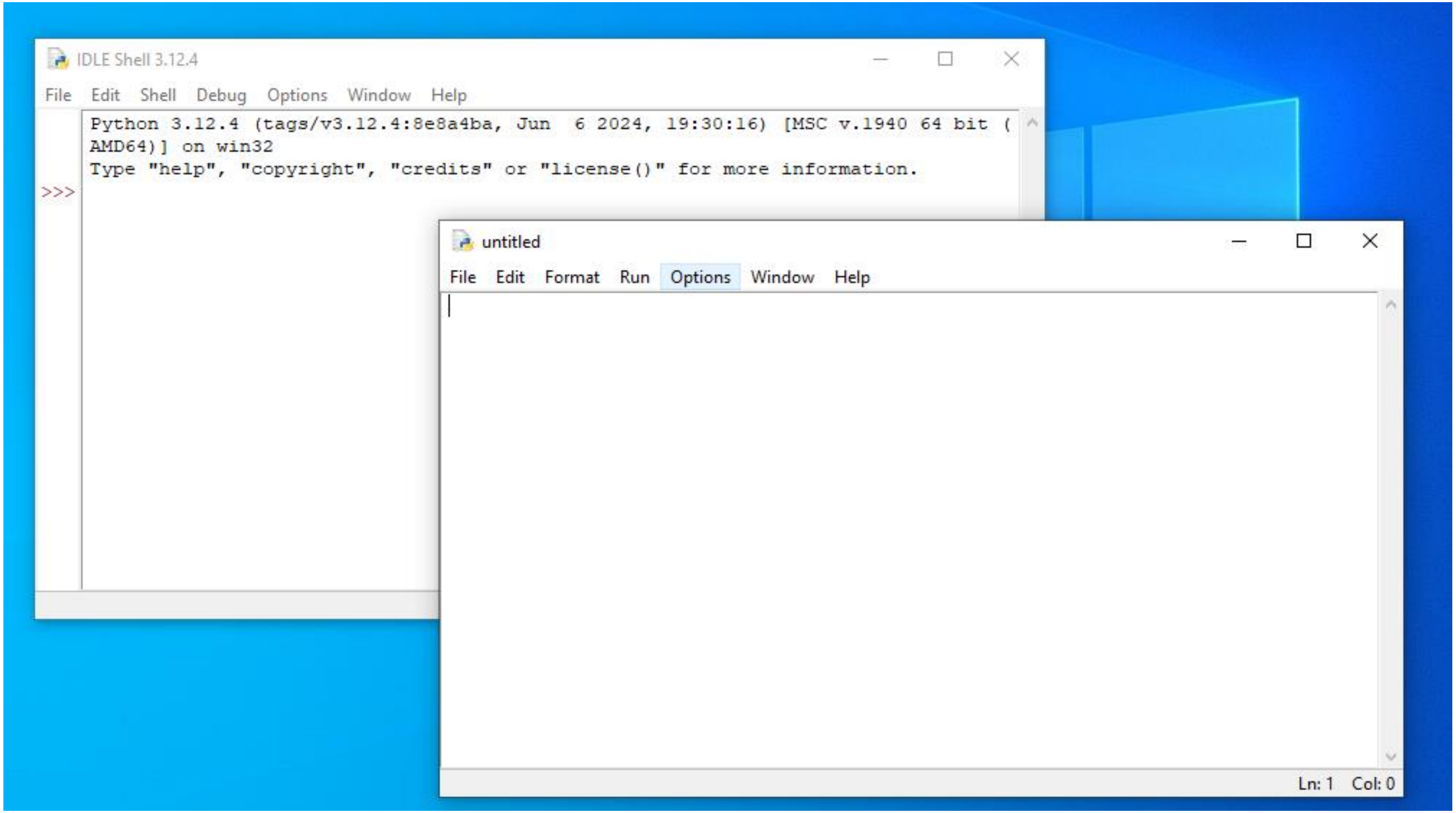
The Interactive Window

IDLE's interactive window contains a Python shell, which is a textual user interface used to interact with the Python language. You can type a bit of Python code into the interactive window and press Enter to immediately see the results

The Editor Window

You'll write your Python files using IDLE's editor window. You can open the editor window by selecting File New File from the menu at the top of the interactive window.

The interactive window stays open when you open the editor window. It displays the output generated by code in the editor window.


IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help

```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

untitled
File Edit Format Run Options Window Help
Ln: 1 Col: 0

What is PVM(Python Virtual Machine) ?

PVM stands for **Python Virtual Machine**. It is a crucial component of the Python runtime environment.

Python Virtual Machine (PVM)

1. Role: The PVM is responsible for executing Python bytecode. When you write Python code, it is first compiled into bytecode, which is a low-level, platform-independent representation of your source code. The PVM then interprets this bytecode and converts it into machine code that the computer's hardware can execute

2. Process:

- **Source Code**: You write Python code in a .py file.
- **Compilation:** The Python interpreter compiles the source code into bytecode (.pyc files).
- **Execution**: The PVM reads the bytecode and executes it, translating it into machine code that the computer can understand and run3.

What is PVM(Python Virtual Machine) ?

3. Advantages:

- **Platform Independence**: Since the bytecode is platform-independent, Python programs can run on any system that has a compatible PVM implementation.
- **Ease of Use**: The PVM handles memory management, garbage collection, and other low-level tasks, allowing developers to focus on writing high-level code.

4. Components:

- **Interpreter**: The component that reads and executes the bytecode.
- **Garbage Collector**: Manages memory allocation and deallocation, ensuring efficient use of resources.
- **Standard Library**: A collection of modules and packages that provide additional functionality for Python programs.

The PVM is an essential part of the Python ecosystem, enabling the language's flexibility and ease of use.

Python Keywords

Keywords are reserved words in Python that have special meanings and cannot be used as identifiers (variable names, function names, etc.).

Here is a list of Python keywords:

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Python Variables

Variables are 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.

Here are some key points about variables in Python:

Variable Naming Rules:

- Must start with a letter (a-z, A-Z) or an underscore (_).
- Can contain letters, digits (0-9), and underscores.
- Case-sensitive (e.g., myVar and myvar are different variables).
- Cannot be a Python keyword.

Assigning Values to Variables:

- You can assign values to variables using the assignment operator (=).
- Example:
- x = 10 name = "Alice" is_active = True

Python Variables

Multiple Assignments:
You can assign values to multiple variables in a single line.
Example:

```
a, b, c = 1, 2, 3
```

Dynamic Typing:
Python is dynamically typed, meaning you don't need to declare the type of a variable.
The type is determined at runtime based on the value assigned.

```
x = 10       # x is an integer
x = "Hello"  # x is now a string
```

Variable Scope:

Variables defined inside a function are local to that function

Variables defined outside any function are global and can be accessed anywhere in the code.

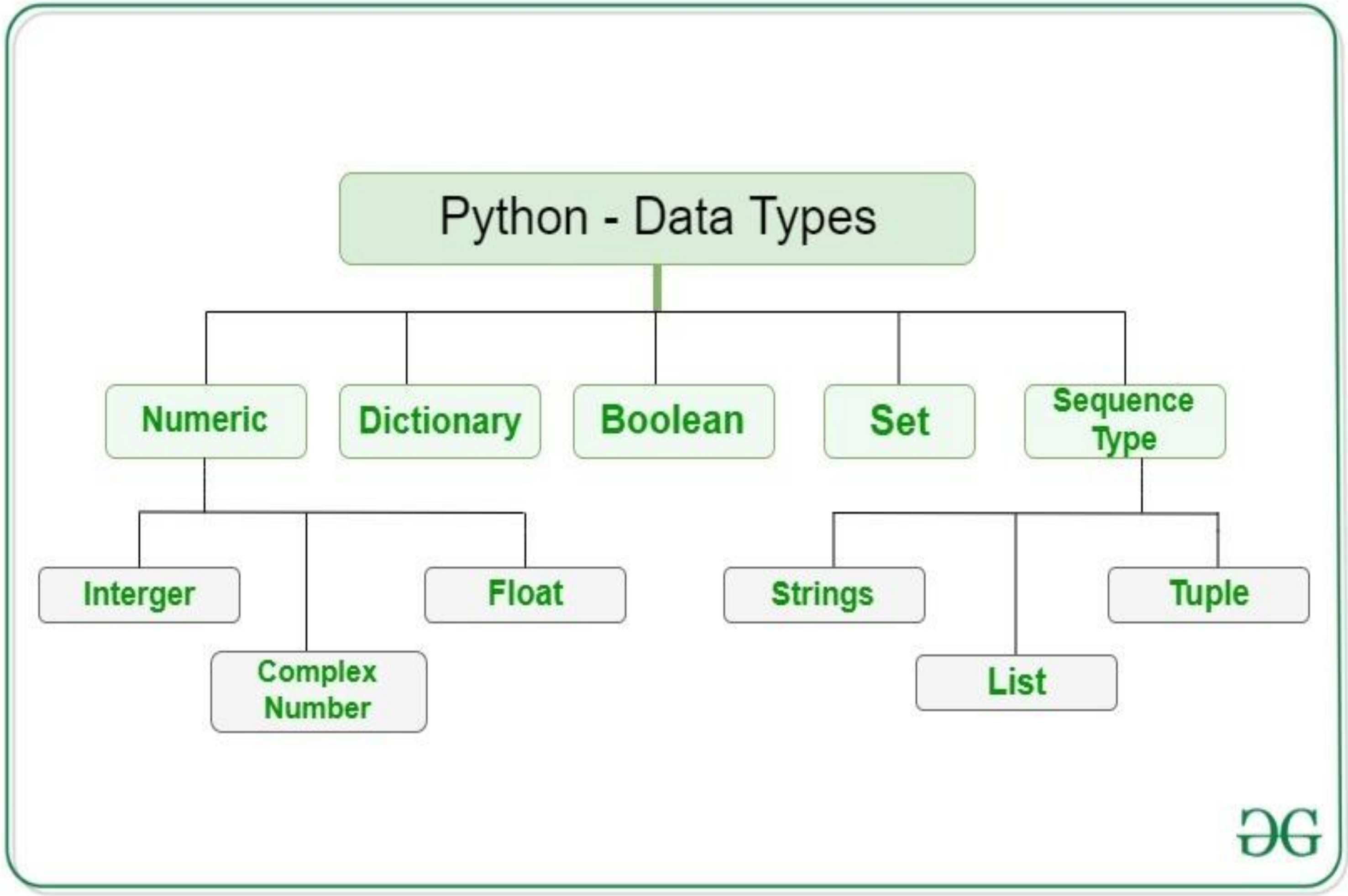
```
def my_function():
    local_var = "I'm local"
            print(local_var)
    ggggg

global_var = "I'm global"
my_function()
print(global_var)
```

Data Types:

Python Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes. The following are the standard or built-in data types in Python:

- ❑ **Numeric – int, float, complex**
- ❑ **Sequence Type – string, list, tuple**
- ❑ **Mapping Type – dict**
- ❑ **Boolean – bool**
- ❑ **Set Type – set, frozenset**
- ❑ **Binary Types – bytes, bytearray, memoryview**



Data Types:

Numeric Data Types in Python

- The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as Python int , Python float , and Python complex classes in Python .

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.

- **Float** – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.

- **Complex Numbers** – A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j* . For example – 2+3j

```
a = 5
print("Type of a: ", type(a))
```

```
Type of a: <class 'int'>
```

```
b = 5.0
print("\nType of b: ", type(b))
```

```
Type of a: <class ‘float'>
```

```
c = 2 + 4j
print("\nType of c: ", type(c))
```

```
Type of a: <class ‘complex'>
```

Data Types:

Sequence Data Types in Python

- The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:
- **Python String** : String, a sequence of characters.
- **Python List**: Ordered, mutable collection of items.
- **Python Tuple**: Ordered, immutable collection of items. (items can not be edited)
- range: Represents a sequence of numbers:

```
r = range(5)  # 0, 1, 2, 3, 4
```

Data Types:

Boolean Type

bool: Represents True or False.

is_active = True

is_logged_in = False

None Type

NoneType : Represents the absence of a value.

result = None

Python Operators

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y It provides the quotient's floor value, which is obtained by dividing the two operands.

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Comparison Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

How to check which memory location python variable or object store?

In Python, you can check the memory location (or address) where a variable or object is stored using the built-in **id() function**.

The id() function returns a unique identifier for the object, which is typically its memory address.

Examples:

```
# Example with integers
a = 10
b = 20
print("Memory address of a:", id(a))
print("Memory address of b:", id(b))

# Example with lists
list1 = [1, 2, 3]
list2 = list1
list3 = [1, 2, 3]
print("Memory address of list1:", id(list1))
print("Memory address of list2:", id(list2))
print("Memory address of list3:", id(list3))

# Example with strings
str1 = "hello"
str2 = "hello"
print("Memory address of str1:", id(str1))
print("Memory address of str2:", id(str2))
```

```
Memory address of a: 140721219369688
Memory address of b: 140721219370008

Memory address of list1: 1856761467008
Memory address of list2: 1856761467008
Memory address of list3: 1856761401280

Memory address of str1: 1856753525232
Memory address of str2: 1856753525232
```

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

How Identity Operators Work?

```
a = [1, 2, 3]
b = a
c = [1, 2, 3]
print(a is b)  # True, because b references the same object as a
print(a is c)  # False, because c is a different object with the same content
---------------------------------------------------------------------------------
x = 10
y = 10
z = 20
print(x is not y)  # False, because x and y reference the same integer object
print(x is not z)  # True, because x and z reference different integer objects
---------------------------------------------------------------------------------
```

x and y reference the same integer object 10, so x is not y returns False.

However, x and z reference different integer objects, so x is not z returns True.

Important Points

- Identity operators compare the memory addresses of the objects, not their values.
- For immutable objects like integers and strings, Python may reuse existing objects with the same value to save memory, which can affect the results of identity comparisons.
- For mutable objects like lists and dictionaries, each object is distinct, even if they have the same content.

Python Membership Operators

Membership operators in Python are used to test whether a value or variable is found in a sequence (such as a string, list, tuple, set, or dictionary).

Operator	**Description**	**Example**
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Membership Operators

```
fruits = ["apple", "banana", "cherry"]
if "banana" in fruits:
    print("Banana is in the list!")



valid_choices = ["yes", "no", "maybe"]
user_input = input("Enter your choice: ")
if user_input in valid_choices:
            print("Valid choice!")
else:
            print("Invalid choice!")

sentence = "The quick brown fox jumps over the lazy dog."
if "fox" in sentence:
            print("The word 'fox' is in the sentence.")
```

Python Bitwise Operators

Bitwise operators in Python are used to perform operations on binary representations of integers. These operators work directly on the bits of the numbers, allowing for efficient manipulation of data at the bit level:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	

Bitwise operations are generally faster than arithmetic operations, making them useful for performance-critical applications, such as graphics processing, cryptography, and network protocols.

Python Bitwise Operators

```
a = 5  # 0101 in binary
b = 3  # 0011 in binary
result = a & b  # 0001 in binary, which is 1 in decimal
print(result)  # Output: 1
```

```
a = 5  # 0101 in binary
b = 3  # 0011 in binary
result = a | b  # 0111 in binary, which is 7 in decimal
print(result)  # Output: 7
```

```
a = 5  # 0101 in binary
result = a << 1  # 1010 in binary, which is 10 in decimal
print(result)  # Output: 10
```

Operators Precedence

Highest at top and lowest at bottom

Operator	Description
()	Parentheses
**	Exponentiation
+x -x ~x	Unary plus, unary minus, and bitwise NOT
* / // %	Multiplication, division, floor division, and modulus
+ -	Addition and subtraction
<< >>	Bitwise left and right shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators
not	Logical NOT
and	AND
or	OR

If two operators have the same precedence, the expression is evaluated from left to right.

Python Input and Output

Python provides several built-in functions for handling input and output operations. Here are the main functions you can use:

Input Functions

input():

This function reads a line from the input (usually from the user) and **returns it as a string.**

Example:

name = input("Enter your name: ")

print("Hello, " + name + "!")

You need to convert it number if you are expecting number from user.

number = int(input("Enter a number: "))

Output Functions

print():

This function prints the specified message to the screen or other standard output device.

```
print("Hello, World!")
print("Hello", "World", 2024)
print("Hello", "World", 2024, sep="-")
```

Using a Custom End Character

By default, print() ends with a newline character. You can change this using the end parameter.

```
print("Hello", end=" ")
print("World!")

name = "Alice"
age = 25
print("Name:" + name)
print("Age:", age)
```

format():

This method formats the specified value(s) and inserts them inside the string's placeholder.

name = "Charlie"

age = 28

country = "Canada"

message = "My name is **{}, I am {} years old, and I live in {}.".format(name, age, country**)

print(message)

Formatted String Literals (f-strings):

Introduced in Python 3.6, f-strings provide a way to embed expressions inside string literals, using curly braces {}.

name = "Alice"

age = 25

print(f"My name is {name} and I am {age} years old.")

Conditional Statements in python

Conditional statements in Python allow you to execute different blocks of code based on certain conditions.

The primary conditional statements in Python are if, elif, and else

The if statement evaluates a condition and executes the block of code inside it if the condition is True.

```
age = 18
if age >= 18:
    print("You are an adult.")
```

The if-else statement provides an alternative block of code to execute if the condition is False.

```
age = 16
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.") # Example of if-else statement
age = 16
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

Conditional Statements in python

The if-elif-else statement allows you to check multiple conditions. The first condition that evaluates to True will have its block of code executed.

```
score = 85
if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: D")
```

Conditional Statements in python

You can nest if statements inside other if statements to check multiple conditions.

```
num = 10
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 not positive.")
```

Using Logical Operators

```
# Example using logical operators
age = 20
has_id = True

if age >= 18 and has_id:
    print("You are allowed to enter.")
else:
    print("You are not allowed to enter.")
```

Conditional Statements in python

Ternary Conditional Operator

Python also supports a shorthand for if-else statements, known as the ternary conditional operator.

```
# Example of ternary conditional operator
age = 18
status = "adult" if age >= 18 else "minor"
print("You are an", status)
```

Control Loops in Python

Control loops in Python are used to repeatedly execute a block of code as long as a condition is met.

There are two main types of control loops in Python: **while loops and for loops**.

while Loop

A while loop repeatedly executes a block of code as long as the given condition is True.

```
count = 0
while count < 5:
    print("Count is:", count)
    count += 1
```

Control Loops in Python

For Loop

A for loop iterates over a sequence (such as a list, tuple, dictionary, set, or string) and executes a block of code for each element in the sequence.

```
for variable in sequence:
    # code block to be executed
```

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print("Fruit:", fruit)
```

Range() in python

The range() function in Python is commonly used in loops to generate a sequence of numbers.

Looping from 0 to 4

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

Looping from 2 to 6

```
for i in range(2, 7):
    print(i)
```

Looping from 0 to 9 with a Step of 2

```
for i in range(0, 10, 2):
    print(i)
```

Looping from 10 to 1 with a Step of -1

```
for i in range(10, 0, -1):
    print(i)
```

Program:

Sum the numbers from 1 to 10 using a loop with range().

```
sum = 0
for i in range(1, 11):
    sum += i
print("The sum is:", sum)
```

Control Statement in Loops in Python

1. break Statement

The break statement is used to exit the loop before it has iterated over all the elements.

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

The loop will terminate when i equals 5.

Continue Statement

The continue statement skips the current iteration of the loop and proceeds to the next iteration.

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

The loop skips printing even numbers.

Else clause in for loop Statement

An else clause can be used with loops. The code inside the else block is executed when the loop exhausts the iterable or the condition becomes false.

```
# Example of else clause in loop
for i in range(5):
    print("i =", i)
else:
    print("Loop completed")
```

Nested Loops

Loops can be nested within other loops, meaning you can have a loop inside a loop.

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