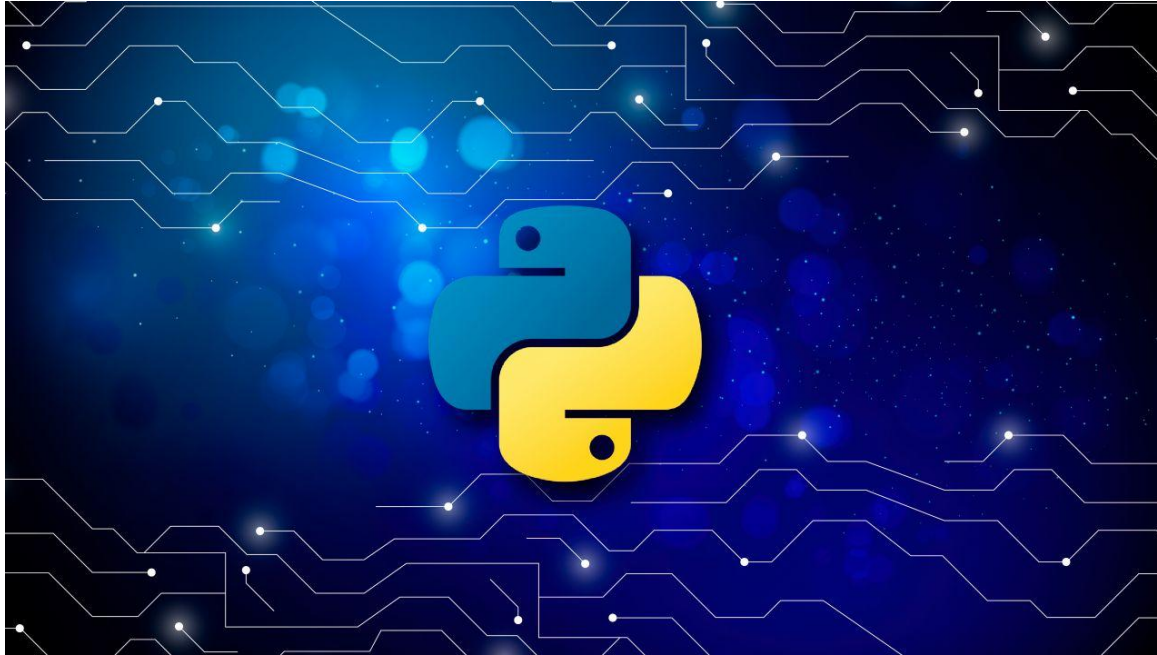


Introduction to Python



Sumedha Kulasekara
Session 1

Introduction

Python is a widely used high-level, interpreted programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. Generally, Python is considered easier to learn than Java, primarily due to its simpler syntax and readability.

What can we do with Python?

Python is used for:

- Web Development: Frameworks like Django, Flask.
- Data Science and Analysis: Libraries like Pandas, NumPy, Matplotlib.
- Machine Learning and AI: TensorFlow, PyTorch, Scikit-learn.
- Automation and Scripting: Automate repetitive tasks.
- Game Development: Libraries like Pygame.
- Web Scraping: Tools like BeautifulSoup, Scrapy.
- Desktop Applications: GUI frameworks like Tkinter, PyQt.
- Scientific Computing: SciPy, SymPy.
- Internet of Things (IoT): MicroPython, Raspberry Pi.
- DevOps and Cloud: Automation scripts and APIs.
- Cybersecurity: Penetration testing and ethical hacking tools.

How is Python different from other programming languages like Java or C++

Python, Java, and C++ are all popular programming languages used for various purposes, but they have distinct characteristics and are suited for different types of development tasks.

1. Syntax and Readability:

Java

```
public class test{  
    public static void main(String[] args)  
    {  
        System.out.println("Python and Java!");  
    }  
}
```

Python

```
print("Python and Java !")
```

2. Typing and Memory Management:

Dynamically typed language where variable types are determined at **runtime**. Python features automatic memory management (garbage collection) which simplifies memory handling.

Java: Statically typed language where variable types are explicitly declared and checked at **compile-time**. Java uses automatic memory management through garbage collection similar to Python.

3. Performance and Execution:

Python Interpreted language with slower execution speed compared to compiled languages like C++ or Java. However, Python's performance can be enhanced by using optimized libraries (Numpy, Pandas)

4. Ecosystem and Libraries:

Python: Has a rich ecosystem of third-party libraries and frameworks for various domains such as web development (Django, Flask), data science (NumPy, Pandas), and machine learning (TensorFlow, PyTorch).

Java: Benefits from a mature ecosystem of libraries and frameworks (e.g., Spring, Hibernate) for enterprise development, web applications, and Android app development.

5. Use Cases and Domains:

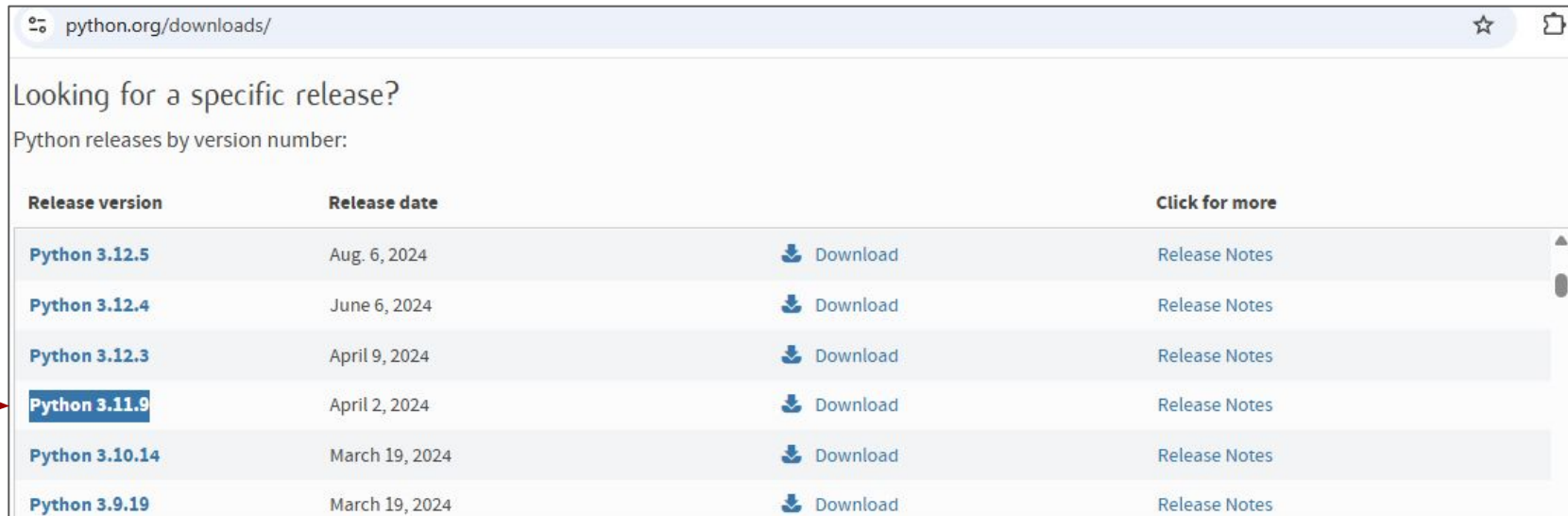
IPython: Ideal for rapid prototyping, scripting, web development, data analysis, and machine learning due to its simplicity and readability.

Java: Commonly used for enterprise applications, server-side development, Android app development, and large-scale systems due to its performance, robustness, and platform independence

Install Python in Windows?

Step 1: Select Version to Install Python

Visit the official page for Python <https://www.python.org/downloads/> on the Windows operating system. Locate a reliable version of Python 3, preferably version Python 3.11.9



python.org/downloads/

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.12.5	Aug. 6, 2024	Download	Release Notes
Python 3.12.4	June 6, 2024	Download	Release Notes
Python 3.12.3	April 9, 2024	Download	Release Notes
Python 3.11.9	April 2, 2024	Download	Release Notes
Python 3.10.14	March 19, 2024	Download	Release Notes
Python 3.9.19	March 19, 2024	Download	Release Notes

Step 2 - Downloading the Python Installer

Once you have downloaded the installer, open the .exe file, such as Python 3.11.9-amd64.exe, by double-clicking to launch the Python installer.

Choose the option to **Install the launcher for all users by** checking the corresponding checkbox, so that all users of the computer can access the Python launcher application. Enable users to run Python from the command line by checking the Add python.exe to PATH checkbox.

Files

Version	Operating System	Description	MD5 Sum	File Size	PGP	Sigstore
Gzipped source tarball	Source release		bfd4d3bfeac4216ce35d7a503bf02d5c	25.3 MB	SIG	.sigstore
XZ compressed source tarball	Source release		22ea467e7d915477152e99d5da856ddc	19.2 MB	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	fa29f456feb6b5c4f52456a8b8ba347b	42.8 MB	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	e8dcd502e34932eebcaf1be056d5cbcd	25.0 MB	SIG	.sigstore
Windows installer (32-bit)	Windows		2a1d1ac2d8a0aa847515f9dd121cbb7	23.8 MB	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	328d93f71cb078965e4cfa2eb2663fa1	24.3 MB	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		6d9aa08531d48fcc261ba667e2df17c4	10.7 MB	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		31e7648158376e92a4463aa6f22a78e1	9.6 MB	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		8611b6aa35483ab1c61d45e0d9f2de0d	10.0 MB	SIG	.sigstore

Install Python in Windows?

Step 3: Running the Executable Installer

After completing the setup, Python will be installed on your Windows system. You will see a successful message.

Step 4: Verify the Python Installation in Windows

Close the window after successful installation of Python. You can check if the installation of Python was successful by using either the command line or the Integrated Development Environment (IDLE).

```
python --version
```

```
C:\Users\User.UDARI>python --version  
Python 3.11.9
```


Install Jupyter Notebook on Windows

Jupyter Notebook is one of the most powerful used among professionals for data science, and machine learning to perform data analysis and data visualization and much more.

Method 1: Using pip

Using PIP with Python is one of the best method to install Jupyter Notebook in your Windows Operating System. Here's how to perform this action in few steps:

Step 1: Check for any Existing Update

Use the following command to update pip (to verify if pip is updated):

```
python -m pip install --upgrade pip
```

Step 2: Install Jupyter Notebook

After updating the pip version, type the following command to install Jupyter:

```
python -m pip install jupyter
```

Step 3: Launching Jupyter

Use the following command to launch Jupyter using:

```
jupyter notebook
```

Now, the Jupyter Notebook will launch automatically in your default web browser.

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, \$, and % within identifiers.

Reserved Words

It cannot be used as constant or variable or any other identifier names.

and	exec	not	assert	finally	or	break
for	pass	class	from	print	continue	
global	raise	def	if	return	del	import
try	elif	in	while	else	is	with
except	lambda	yield				

Creating Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

```
x= 5  
y= "John"  
print(x)  
print(y)
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

#Legal variable names:

```
myvar = "John"  
my_var = "John"  
_my_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"
```

#Illegal variable names:

```
2myvar = "John"  
my-var = "John"  
my var = "John"
```

Assign Value to Multiple Variables

```
x, y, z = "Orange", "Banana", "Cherry"  
x = y = z = "Orange"
```

Output Variables

The Python print statement is often used to output variables. To combine text and a variable, Python uses the + character: Ex

```
x = "awesome"  
print("Python is " + x)
```

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Computer", "Science", "Programme"  
print(x)  
print(y)  
print(z)
```

One Value to Multiple Variable

And you can assign the same value to multiple variables in one line:

```
x = y = z = "Computer"  
print(x)  
print(y)  
print(z)
```

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called unpacking.

```
names= ["John", "Bruce", "Ryan"]  
x, y, z = names  
print(x)  
print(y)  
print(z)
```

Output Variables

The Python print() function is often used to output variables.

```
x = "Computer Science with AI"  
print(x)
```

Global Variables

Variables that are created outside of a function are known as global variables. Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"  
def myfunc():  
    x = "fantastic"  
    print("Python is " + x)
```

```
myfunc()  
print("Python is " + x)
```

What will be the printed result?

```
x = 'Computer'  
def myfunc():  
    x = 'Science'  
myfunc()  
print('Output' + x)
```

The global Keyword

When you create a variable inside a function, that variable is local, and can only be used inside that function. To create a global variable inside a function, you can use the global keyword.

```
x = "awesome"
def myfunc():
    global x
    x = "fantastic"

myfunc()
print("Python is " + x)
```


Python Data Types

data type is an important concept in programming languages.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Getting the Data Type

```
x = 5  
print(type(x))
```

```
y = "hello"  
print(type(y))
```

```
z = [1, 2, 3]  
print(type(z))
```

```
a = 3.14  
print(type(a))
```

```
b = True  
print(type(b))
```

1. Numeric Types

int (Integer)

Whole numbers, positive or negative, without decimals.

```
x = 10 # Integer  
y = -5 # Negative integer  
print(type(x))
```

float (Floating-Point Number)

Numbers with decimal points.

```
x = 10.5 # Float  
y = -3.14 # Negative float  
print(type(x))
```

Complex (Complex Number)

Numbers with a real and imaginary part (written as $a + bj$).

```
z = 3 + 4j  
print(type(z))  
print(z.real) # Output: 3.0  
print(z.imag) # Output: 4.0
```

2. Sequence Types

list (Mutable Ordered Collection)

A collection of items that can be changed (mutable).

```
my_list = [1, 2, 3, "hello"]  
my_list.append(4)  
print(my_list) # Output: [1, 2, 3, 'hello', 4]
```

tuple (Immutable Ordered Collection)

Like a list but cannot be changed (immutable).

```
my_tuple = (1, 2, 3, "hello")  
# my_tuple[0] = 10 # Error: Tuples are immutable  
print(my_tuple[1]) # Output: 2
```

Range (Sequence of Numbers)

Used for generating a sequence of numbers.

```
r = range(5) # 0 to 4  
print(list(r)) # Output: [0, 1, 2, 3, 4]
```

```
r = range(2, 7)  
print(list(r)) # [2, 3, 4, 5, 6]
```

```
r = range(1, 10, 2)  
print(list(r)) # [1, 3, 5, 7, 9]
```

3. Mapping Type

dict (Dictionary - Key-Value Pairs)

Stores data in key-value pairs.

```
student = {"name": "Alice", "age": 20}  
print(student["name"]) # Output: Alice  
student["age"] = 21 # Changing a value
```

- Difference:

Unlike lists, dictionaries store key-value pairs.

4. Set Types

set (Mutable, Unordered Collection of Unique Elements)

```
my_set = {1, 2, 3, 3} # Duplicates are removed
print(my_set) # Output: {1, 2, 3}
my_set.add(4)
print(my_set)
```

frozenset (Immutable Set)

Like a set but cannot be modified.

```
fs = frozenset([1, 2, 3,3])
print(fs)
fs.add(4) # Error: Frozensets are immutable
```

5. Boolean Type

bool (True or False Values)

```
x = True  
y = False  
print(5 > 3) # Output: True  
print(10 == 5) # Output: False
```

6. Binary Types

bytes (Immutable Sequence of Bytes)

```
b = b"hello"  
print(b) # Output: b'hello'
```

'h' → 104

'e' → 101

'l' → 108

'l' → 108

'o' → 111

- The `b""` prefix creates a bytes object, which is an immutable sequence of bytes.
- Each character in `"hello"` is stored as its ASCII equivalent:

(ASCII (American Standard Code for Information Interchange) is a system that assigns a unique number (code) to each character (letters, digits, symbols) used in computing.)

The variable `b` now holds: `b'hello'`

This means the text is stored in byte form.

bytearray (Mutable Sequence of Bytes)

- bytearray is mutable, meaning we can change its elements.
- ASCII values map numbers to characters (65 → A, 68 → D).

```
ba = bytearray([65, 66, 67])  
ba[0] = 68  
print(ba) # Output: bytearray(b'DBC')
```

memoryview (Views on Binary Data Without Copying)

```
b = bytes([65, 66, 67])  
m = memoryview(b)  
print(m[0]) # Output: 65
```

DIFFERENCES

- bytes: Immutable
- bytearray: Mutable
- memoryview: Efficient access to binary data

7. None Type

NoneType (Represents Absence of a Value)

```
x = None  
print(x is None) # Output: True
```

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- ❑ Equals: **a == b**
- ❑ Not Equals: **a != b**
- ❑ Less than: **a < b**
- ❑ Less than or equal to: **a <= b**
- ❑ Greater than: **a > b**
- ❑ Greater than or equal to: **a >= b**

These conditions can be used in several ways, most commonly in "if statements" and loops.

if

An "if statement" is written by using the if keyword.

```
a = 25
b = 26
if b > a:
    print("b is greater than a")
```

NOTE :-

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Else

The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 26
b = 22
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

Elif

is Python's way of saying "if the previous conditions were not true, then try this condition".

```
a = 25
b = 20
if b > a:
    print("b is greater than a")
elif b < a:
    print("a is greater than b")
else:
    print("a and b are equal")
```

Python has two primitive loop commands:

1) While loop

```
i = 1
while i < 8:
    print(i)
    i += 1
```

Note: remember to increment i, or else the loop will continue forever.

```
i = 1
while i < 8:
    print(i)
    if i == 3:
        break
    i += 1
```

```
i = 1
while i < 8:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

- Using break statement we can stop the loop even if the while condition is true
- With the continue statement we can stop the current iteration, and continue with the next:
- With the else statement we can run a block of code once when the condition no longer is true:

Python has two primitive loop commands:

2) A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
Subjects= ["Maths", "Physics", "Chemistry"]  
for x in Subjects:  
    print(x)
```

Try: -

```
for x in "Computer Science":  
    print(x)
```

```
for x in range(2,12 ):  
    print(x)
```

```
for x in range(2, 50, 5):  
    print(x)
```

Nested Loops

A nested loop is a loop inside a loop.

```
Subjects= ["Maths", "Physics", "Chemistry"]  
Semesters= ["sem1", "sem2", "sem3"]
```

```
for x in Subjects:  
    for y in Semesters:  
        print(x, y)
```

Rounding numbers

```
number = 3.14159
```

```
rounded_number = round(number, 2)
```

```
print(rounded_number)
```

```
print(round(4.7)) # Rounds up to 5
```

```
print(round(4.2)) # Rounds down to 4
```

```
number = 3.56789
```

```
print(round(number, 1)) # Rounds to 1 decimal place
```

```
print(round(number, 3)) # Rounds to 3 decimal places
```

```
import math
```

```
print(math.ceil(4.2)) # Rounds up to 5
```

```
print(math.ceil(4.7)) # Also rounds up to 5
```

```
print(math.floor(4.2)) # Rounds down to 4
```

```
print(math.floor(4.7)) # Also rounds down to 4
```


Python Arithmetic Operators

Arithmetic operators are symbols used to perform mathematical operations on numerical values.

Arithmetic operators include addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).

```
val1 = 2
```

```
val2 = 3
```

```
# using the addition operator
```

```
res = val1 + val2
```

```
print(res)
```

```
# using the subtraction operator
```

```
res = val1 - val2
```

```
print(res)
```

```
# using the multiplication operator
```

```
res = val1 * val2
```

```
print(res)
```

Float division - The quotient returned by this operator is always a float number, no matter if two numbers are integers. For example:

```
print(5/5)
print(10/2)
print(-10/2)
print(20.0/2)
```

Integer division(Floor division)

The quotient returned by this operator is dependent on the argument being passed. If any of the numbers is float, it returns output in float. It is also known as Floor division because, if any number is negative, then the output will be floored. For example:

```
print(10//3)
print (-5//2)
print (5.0//2)
print (-5.0//2)
```

Modulus Operator- The % in Python is the modulus operator. It is used to find the remainder when the first operand is divided by the second.

```
res = 10 % 7  
print(res)
```

Exponentiation Operator- In Python, ** is the exponentiation operator. It is used to raise the first operand to the power of the second.

```
val1 = 2  
val2 = 3
```

```
# using the exponentiation operator  
res = val1 ** val2  
print(res)
```

Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

1)

```
def my_function():  
    print("First_Function")
```

```
my_function()
```

2)

```
def my_function(fname, lname):  
    print(fname + " " + lname)
```

```
my_function("Jane", "Mackwills")
```

3)
def my_function(city = "Kandy"):
 print("I am from " + city)

my_function("Colombo")
my_function("Galle")
my_function()
my_function("Mathale")

4)
def my_function(subjects):
 for x in subjects:
 print(x)

subjects= ["Science", "English", "History"]

my_function(subjects)

5)
def my_function(x):
 return 10* x

print(my_function(10))
print(my_function(30))
print(my_function(2))

Exercise

- 1) Write a function `is_even_or_odd(number)` that takes an integer and returns "Even" if the number is even and "Odd" if the number is odd.
- 2) Write a function `sum_of_numbers(numbers)` that takes a list of numbers and returns the sum of all the numbers in the list.
- 3) Write a function `find_largest(numbers)` that takes a list of numbers and returns the largest number in the list.
- 4) Write a function `merge_lists(list1, list2)` that takes two lists and returns a single list that contains the elements from both lists.
- 5) Write a function `area_of_circle(radius)` that takes the radius of a circle as input and returns the area of the circle. Use the formula $\text{Area} = \pi \times (r^2)$

Answers

1)

```
def is_even_or_odd(number):  
    if number % 2 == 0:  
        return "Even"  
    else:  
        return "Odd"
```

#usage

```
print(is_even_or_odd(4)) # Output: Even  
print(is_even_or_odd(7)) # Output: Odd
```

2)

```
def sum_of_numbers(numbers):  
    return sum(numbers)
```

#usage

```
numbers = [1, 2, 3, 4, 5]  
print(sum_of_numbers(numbers))
```

3)

```
def find_largest(numbers):  
    return max(numbers)
```

#usage

```
numbers = [10, 20, 30, 50, 40]  
print(find_largest(numbers)) # Output: 50
```

Answers

4)

```
def merge_lists(list1, list2):  
    return list1 + list2
```

usage

```
list1 = [1, 2, 3]
```

```
list2 = [4, 5, 6]
```

```
print(merge_lists(list1, list2))
```

5)

```
import math
```

```
def area_of_circle(radius):
```

```
    return math.pi * radius ** 2
```

Example usage

```
print(area_of_circle(5)) # Output: 78.53981633974483
```


Python libraries : are collections of pre-written code and functions that extend the capabilities of the Python programming language, providing tools and modules for various tasks, making it easier for developers to work on specific tasks.

Examples of popular Python libraries:

- NumPy: A fundamental library for numerical computing, providing support for arrays, matrices, and mathematical functions.
- Pandas: A library for data analysis and manipulation, offering data structures like DataFrames and Series.
- Matplotlib: A library for creating static, interactive, and animated visualizations in Python.

Python Dates

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

```
import datetime
x = datetime.datetime.now()
print(x)
```

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

Create a date object:

```
import datetime
x = datetime.datetime(2020, 5, 17)
print(x)
```

Maths

Built-in Math Functions

The `min()` and `max()` functions can be used to find the lowest or highest value in an iterable:

```
x = min(5, 15,22,10, 25)
```

```
y = max(5, 15,22,10, 25)
```

```
print(x)
```

```
print(y)
```

The `abs()` function returns the absolute (positive) value of the specified number:

```
x = abs(-7.25)
```

```
print(x)
```

The `pow(x, y)` function returns the value of `x` to the power of `y` (x^y).

```
x = pow(10, 2)
```

```
print(x)
```

Python has also a built-in module called **math**, which extends the list of mathematical functions.

```
import math  
x = math.sqrt(225)  
print(x)
```

```
y= math.pi  
print(y)
```

PIP?

pip is the package installer for Python. You can use pip to install packages from the Python Package Index and other indexes.

What is a Package?

A package contains all the files you need for a module.

Modules are Python code libraries you can include in your project.

Check if PIP is Installed

pip --version

```
C:\Users\User.UDARI>pip --version
pip 24.0 from C:\Users\User.UDARI\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip (python 3.11)
```

Download a package named "camelcase":

pip install camelcase

Jupyter Notebook

!pip install camelcase

```
C:\Users\User.UDARI>pip install camelcase
Collecting camelcase
  Downloading camelcase-0.2.tar.gz (1.3 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: camelcase
  Building wheel for camelcase (setup.py) ... done
  Created wheel for camelcase: filename=camelcase-0.2-py3-none-any.whl size=1900 sha256=9e01eb7dc529ac655c0dc336d35ef036
  Stored in directory: c:\users\user.udari\appdata\local\programs\python\python311\lib\site-packages\pip\Cache\wheels\059eb6739165b
Successfully built camelcase
Installing collected packages: camelcase
Successfully installed camelcase-0.2
```

User Input

means we are able to ask the user for input.

```
username = input("Enter username:")  
print("Username is: " + username)
```

Exercises

- 1) Ask the user for their name and age. Print a personalized greeting. If the user is younger than 18, print "You are a minor"; else, "You are an adult".
- 2) Build a calculator that performs addition, subtraction, multiplication, and division. Let the user choose the operation and input two numbers.

Answers

1)

```
name = input("Enter your name: ")
```

```
age = int(input("Enter your age: "))
```

```
print(f"Hello, {name}!")
```

```
if age < 18:
```

```
    print("You are a minor.")
```

```
else:
```

```
    print("You are an adult.")
```


2)

```
def calculator(a, b, op):
```

```
    if op == '+':
```

```
        return a + b
```

```
    elif op == '-':
```

```
        return a - b
```

```
    elif op == '*':
```

```
        return a * b
```

```
    elif op == '/':
```

```
        return a / b
```

```
    else:
```

```
        return "Invalid operator"
```

```
while True:
```

```
    op = input("Choose operation (+, -, *, /) or 'q' to quit: ")
```

```
    if op == 'q':
```

```
        break
```

```
    x = float(input("Enter first number: "))
```

```
    y = float(input("Enter second number: "))
```

```
    result = calculator(x, y, op)
```

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

2)

```
def calculator(a, b, op):
```

```
    if op == '+':
```

```
        return a + b
```

```
    elif op == '-':
```

```
        return a - b
```

```
    elif op == '*':
```

```
        return a * b
```

```
    elif op == '/':
```

```
        return a / b
```

```
    else:
```

```
        return "Invalid operator"
```

```
# User input for operation
```

```
op = input("Choose operation (+, -, *, /) or 'q' to quit: ")
```

```
if op != 'q':
```

```
    # If the user doesn't want to quit, proceed with input and calculation
```

```
    x = float(input("Enter first number: "))
```

```
    y = float(input("Enter second number: "))
```

```
    result = calculator(x, y, op)
```

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

```
else:
```

```
    print("Goodbye!")
```

Classes/Objects

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

```
class vehicle:  
    def __init__(self, type, brand):  
        self.type = type  
        self.brand = brand
```

```
p1 = vehicle("Van", "Toyota")
```

```
print(p1.type)  
print(p1.brand)
```

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object. Let us create a method in the vehicle class:

```
class Vehicle:
    def __init__(self, type, brand):
        self.type = type
        self.brand = brand

    def describe(self):
        print("This is a " + self.brand + " " + self.type)

# Creating object
v1 = Vehicle("Van", "Toyota")

# Calling method
v1.describe()
```

The **self** parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It **does not have to be named self**, you can call it whatever you like, but it has to be the first parameter of any function in the class:

```
class Vehicle:
    def __init__(myobject, type, brand):
        myobject.type = type
        myobject.brand = brand

    def describe(abc):
        print("This is a " + abc.brand + " " + abc.type)

# Creating object
v1 = Vehicle("Van", "Toyota")

# Calling method
v1.describe()
```

Exercise

Create a class called BankAccount that simulates a simple bank account.

Your class should include:

Attributes:

owner – the name of the account holder

balance – the current balance (default should be 0)

Methods:

deposit(self, amount)

➤ Adds the given amount to the balance

withdraw(self, amount)

➤ Subtracts the amount from the balance if sufficient funds are available

➤ If not, print "Insufficient balance."

show_balance(self)

➤ Prints the current balance in the format: "Balance: <amount>"

```
class BankAccount:
    def __init__(self, owner):
        self.owner = owner
        self.balance = 0

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Insufficient balance.")

    def show_balance(self):
        print(f"Balance: {self.balance}")
```

```
# Example usage
acc = BankAccount("John")
acc.deposit(500)
acc.show_balance()
acc.withdraw(200)
acc.show_balance()
```