Python

Python was created by Guido van Rossum in 1989.He named it after the comedy show Monty Python's Flying Circus. The first version was released in 1991, and it quickly gained popularity due to its simplicity and readability. Over the years, Python grew into one of the most popular programming languages, used in fields like web development, data science, and AI.

In [1]:
```python
# to print Hello World
print("Hello World!")
```
```
Hello World!
```

In [2]:
```python
print("Hello Python")
print('Hello Python')
```
```
Hello Python
Hello Python
```

Data Type:

A data type in Python refers to the type of value a variable can hold. It determines what kind of operations can be performed on that value.

The basic data types in Python include:

int: Integer numbers (whole numbers) Example: 5, -3, 100

float: Floating-point numbers (decimal numbers) Example: 3.14, -0.001, 2.0

str: Strings (sequences of characters) Example: "Hello", 'Python'

bool: Boolean values representing True or False Example: True, False

Variable

A variable is a name that refers to a value or data stored in memory. It allows you to store and manipulate data. You can assign different data types to variables, such as integers, floats, and booleans.

In [3]:
```python
age = 25  # Variable 'age' holds an integer value 25
print(age)

name = "John"  # Variable 'name' holds a string value "John"
print(name)

price = 19.99  # Variable 'price' holds a floating-point value 19.99
print(price)

is_student = True  # Variable 'is_student' holds a boolean value True
print(is_student)
```
```
25
John
19.99
True
```

Type Casting

Type casting in Python refers to converting one data type into another. Python provides two types of type casting:

1. Implicit Type Casting (Automatic)
2. Explicit Type Casting (Manual)

In [4]:
```python
# Implicit Type Casting (Automatic)
x = 10    # Integer
y = 2.5   # Float

result = x + y  # Python automatically converts `x` to float
print(result)   # Output: 12.5
print(type(result))  # Output: <class 'float'>
```
```
12.5
<class 'float'>
```

In [5]:
```python
# Example 1: Converting float to int (Decimal part is truncated)
a = 7.8
a_int = int(a)
print(a_int)
print(type(a_int))

# Example 2: Converting string to float
s = "45.67"
s_float = float(s)
print(s_float)
print(type(s_float))

# Example 3: Converting int to string
num = 100
num_str = str(num)
print(num_str)
print(type(num_str))

# Example 4: Converting boolean to int (True = 1, False = 0)
b = False
b_int = int(b)
print(b_int)
print(type(b_int))

# Example 5: Converting int to boolean (0 = False, any nonzero number = True)
x = 10
x_bool = bool(x)
print(x_bool)
print(type(x_bool))
```
```
7
<class 'int'>
45.67
<class 'float'>
100
<class 'str'>
0
<class 'int'>
True
<class 'bool'>
```

String

In [6]:
```python
a = "Hello"    # Single line String

b = 'Hello'    # Single line String

c = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""     # Multiline String

d = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''# Multiline String
print(a)
print(b)
print(c)
print(d)
```
```
Hello
Hello
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Slicing String

Slicing in Python allows you to extract a portion of a sequence

In [7]:
```python
text = "Python Programming"

# Get the first 6 characters (step = 1)
print(text[0:6:1])   # Output: Python

# Get characters from index 7 to the end (step = 1)
print(text[7::1])    # Output: Programming

# Get characters from start to index 9 (step = 1)
print(text[:10:1])   # Output: Python Pro

# Get every third character (step = 3)
print(text[::3])     # Output: Ph oai

# Get every second character starting from index 1 (step = 2)
print(text[1::2])    # Output: yhnPormig

# Get characters from index 3 to 12 with a step of 2
print(text[3:12:2])  # Output: h rga

# Reverse the string (step = -1)
print(text[::-1])    # Output: gnimmargorP nohtyP

# Reverse every second character (step = -2)
print(text[::-2])    # Output: gimro Phy

# Get last 5 characters (step = 1)
print(text[-5:])     # Output: mming

# Get the entire string (step = 1)
print(text[:])       # Output: Python Programming
```
```
Python
Programming
Python Pro
Ph oai
yhnPormig
hnPor
gnimmargorP nohtyP
gimroPnhy
mming
Python Programming
```

String Methods

Python provides many built-in string methods to manipulate and process strings efficiently.

In [8]:
```python
# lower() - Convert to Lowercase
text = "HELLO WORLD"
print(text.lower())
```
```
hello world
```

In [9]:
```python
# upper() - Convert to Uppercase
text = "hello world"
print(text.upper())
```
```
HELLO WORLD
```

In [10]:
```python
# strip() - Remove Leading & Trailing Spaces
text = "  Python Programming  "
print(text.strip())
```
```
Python Programming
```

In [11]:
```python
# replace(old, new) - Replace Substring
text = "I love Python"
print(text.replace("Python", "Java"))  # Output: I love Java
```
```
I love Java
```

In [12]:
```python
# split(separator) - Split into a List
text = "Apple, Banana, Cherry"
fruits = text.split(", ")
print(fruits)  # Output: ['Apple', 'Banana', 'Cherry']
```
```
['Apple', 'Banana', 'Cherry']
```

In [13]:
```python
# join(iterable) - Join List into String
words = ["Python", "is", "awesome"]
sentence = " ".join(words)
print(sentence)  # Output: Python is awesome
```
```
Python is awesome
```

In [14]:
```python
# capitalize() - Capitalize First Letter
text = "hello world"
print(text.capitalize())  # Output: Hello world
```
```
Hello world
```

In [15]:
```python
# title() - Capitalize First Letter of Each Word
text = "hello world"
print(text.title())  # Output: Hello World
```
```
Hello World
```

In [16]:
```python
# count(substring) - Count Occurrences of a Substring
text = "apple mango grapes apple"
print(text.count("apple"))  # Output: 2
```
```
2
```

In [17]:
```python
# startswith(substring) - Check if String Starts with Given Substring
text = "Python Programming"
print(text.startswith("Python"))  # Output: True
```
```
True
```

In [18]:
```python
# endswith(substring) - Check if String Ends with Given Substring
text = "Python Programming"
print(text.endswith("Java"))  # Output: False
```
```
False
```

In [19]:
```python
# find(substring) - Find Index of Substring
text = "Hello, welcome to Python"
print(text.find("Python"))  # Output: 18
print(text.find("Java"))    # Output: -1 (not found)
```
```
18
-1
```

In [20]:
```python
# isalpha() - Check if All Characters are Letters
text = "HelloWorld"
print(text.isalpha())  # Output: True

text2 = "Hello123"
print(text2.isalpha())  # Output: False
```
```
True
False
```

In [21]:
```python
# isdigit() - Check if All Characters are Digits
text = "12345"
print(text.isdigit())  # Output: True

text2 = "123abc"
print(text2.isdigit())  # Output: False
```
```
True
False
```

These methods return new values and do not modify the original string because strings in Python are immutable

In [22]:
```python
# Concatenate String
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```
```
Hello World
```

Escape characters

In [23]:
```python
# Escaping a single quote
print('I\'ll call you later')

# Escaping a double quote
print("Welcome to \"Aptech\" Defence")
# Newline character (\n)
print("Hello \nStudents")

# Tab character (\t)
print("Hello \tStudents")

# Carriage return (\r)
print("no__benefits__of_study\rHuge_benefits")
# Backspace (\b)
print("Hello \bStudents")
```
```
I'll call you later
Welcome to "Aptech" Defence
Hello
Students
Hello   Students
Huge_benefits
Hello ▯Students
```

Python User Input

In [25]:
```python
# Basic String Input
name = input("Enter your name: ")
print("Hello, " + name + "!")
```
```
Enter your name: John
Hello, John!
```

In [26]:
```python
# Taking Integer Input
age = int(input("Enter your age: "))
print("You are", age, "years old.")
```
```
Enter your age: 14
You are 14 years old.
```

In [27]:
```python
# Taking Float Input
height = float(input("Enter your height in meters: "))
print("Your height is", height, "meters.")
```
```
Enter your height in meters: 2.37
Your height is 2.37 meters.
```

In [28]:
```python
# Taking Multiple Inputs (Space-Separated)
x, y = input("Enter two numbers separated by space: ").split()
print("You entered:", x, "and", y)
```
```
Enter two numbers separated by space: 10 22
You entered: 10 and 22
```

Operators in Python

Operators are symbols or special characters in programming that perform specific operations on values or variables.

1. Arithematic Operators
2. Comparison Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

1. Arithematic Operators

Arithmetic operators in Python are symbols that perform basic mathematical operations on numbers (integers or floating-point).

In [29]:
```python
# 1. Addition (+): Adds two numbers together.
a = 5
b = 3
result = a + b
print(result)
```
```
8
```

In [30]:
```python
# 2. Subtraction (-): Subtracts one number from another.
a = 10
b = 5
result = a - b
print(result)
```
```
5
```

In [31]:
```python
# 3. Multiplication (*): Multiplies two numbers.
a = 7
b = 3
result = a * b
print(result)
```
```
21
```

In [32]:
```python
# 4. Division (/): Divides one number by another and returns a float.
a = 30
b = 2
result = a / b
print(result)
```
```
15.0
```

In [33]:
```python
# 5. Modulo (%): Returns the remainder of the division.
a = 5
b = 2
result = a % b
print(result)
```
```
1
```

In [34]:
```python
# 6. Floor Division (//): Divides one number by another and returns the quotient without the remainder
# (rounded down to the nearest integer).
a = 5
b = 2
result = a // b
print(result)
```
```
2
```

In [35]:
```python
# 7. Exponentiation (**): Raises one number to the power of another.
a = 2
b = 3
result = a ** b
print(result)
```
```
8
```

2. Comparison Operator

Comparison operators in Python are used to compare two values or expressions and return a Boolean value (True or False) based on the result of the comparison.

In [36]:
```python
# 1. == (Equal to): Compares if two values are equal.
a = 5
b = 5
print(a == b)
```
```
True
```

In [37]:
```python
# 2. != (Not equal to): Compares if two values are not equal.
a = 5
b = 3
print(a != b)
```
```
True
```

In [38]:
```python
# 3. > (Greater than): Compares if the left value is greater than the right value.
a = 5
b = 3
print(a > b)
print(a < b)
```
```
True
False
```

In [39]:
```python
# 4. < (Less than): Compares if the left value is less than the right value.
a = 5
b = 7
print(a < b)
print(a > b)
```
```
True
False
```

In [40]:
```python
# 5. >= (Greater than or equal to): Compares if the left value is greater than or equal to the right value.
a = 5
b = 5
print(a >= b)
```
```
True
```

In [41]:
```python
# 6. <= (Less than or equal to): Compares if the left value is less than or equal to the right value.
a = 3
b = 5
print(a <= b)
```
```
True
```

3. Logical Operators

Logical operators in Python are used to combine multiple conditional statements or expressions. They help you evaluate multiple conditions and return a Boolean value (True or False).

In [42]:
```python
# and: Returns True if both conditions are True. If either condition is False, it returns False.
a = 5
b = 3
c = 7
print(a > b and c > a)
print(a > b and c < a)
```
```
True
False
```

In [43]:
```python
# or: Returns True if at least one of the conditions is True. It only returns False when both conditions are False.
a = 5
b = 3
c = 7
print(a > b or c < a)
print(a < b or c < a)
```
```
True
False
```

In [44]:
```python
# not: Reverses the Boolean value of the condition. If the condition is True, it returns False;
# if the condition is False, it returns True.
a = 5
b = 3
print(not(a > b))
print(not(a < b))
print(not(a == b))
print(not(a != b))
```
```
False
True
True
False
```

4. Assigment Operators

Assignment operators in Python are used to assign values to variables.

In [45]:
```python
# 1. = (Simple Assignment): This operator is used to assign a value to a variable.
a = 10  # Assigns 10 to variable a
print(a)
```
```
10
```

In [46]:
```python
# 2. += (Add and Assign): This operator adds the right-hand value to the left-hand variable and then assigns the
# result back to the left-hand variable.
a = 10
a += 5  # Equivalent to a = a + 5
print(a)
```
```
15
```

In [47]:
```python
# 3. -= (Subtract and Assign): This operator subtracts the right-hand value from the left-hand variable and
# then assigns the result back to the left-hand variable.
a = 10
a -= 3  # Equivalent to a = a - 3
print(a)
```
```
7
```

In [48]:
```python
# 4. *= (Multiply and Assign): This operator multiplies the left-hand variable by the right-hand value and
# then assigns the result back to the left-hand variable.
a = 10
a *= 2  # Equivalent to a = a * 2
print(a)
```
```
20
```

In [49]:
```python
# 5. /= (Divide and Assign): This operator divides the left-hand variable by the right-hand value and
# then assigns the result back to the left-hand variable (returns a float).
a = 10
a /= 2  # Equivalent to a = a / 2
print(a)
```
```
5.0
```

In [50]:
```python
# 6. %= (Modulo and Assign): This operator calculates the remainder of dividing the left-hand variable
# by the right-hand value and assigns the result back to the left-hand variable.
a = 10
a %= 3  # Equivalent to a = a % 3
print(a)
```
```
1
```

In [51]:
```python
# 7. //= (Floor Division and Assign): This operator divides the left-hand variable by the right-hand value and
# assigns the result back to the left-hand variable, rounding down to the nearest integer.
a = 10
a //= 3  # Equivalent to a = a // 3
print(a)
```
```
3
```

In [52]:
```python
# 8. **= (Exponentiation and Assign): This operator raises the left-hand variable to the power of the right-hand value and
# assigns the result back to the left-hand variable.
a = 2
a **= 3  # Equivalent to a = a ** 3
print(a)
```
```
8
```

5. Bitwise Operators

Bitwise operators in Python are used to perform operations on the binary representations of integers.

In [53]:
```python
# 1. & (Bitwise AND): Compares each bit of two numbers and returns 1 if both bits are 1, otherwise returns 0.
a = 5  # 0101 in binary
b = 3  # 0011 in binary
result = a & b  # Bitwise AND
print(result)  # Output: 1 (0001 in binary)
```
```
1
```

In [54]:
```python
# 2. | (Bitwise OR): Compares each bit of two numbers and returns 1 if at least one of the bits is 1,
# otherwise returns 0.
a = 5  # 0101 in binary
b = 3  # 0011 in binary
result = a | b  # Bitwise OR
print(result)  # Output: 7 (0111 in binary)
```
```
7
```

In [55]:
```python
# 3. ^ (Bitwise XOR - Exclusive OR): Compares each bit of two numbers and returns 1 if the bits are different,
# otherwise returns 0.
a = 5  # 0101 in binary
b = 3  # 0011 in binary
result = a ^ b  # Bitwise XOR
print(result)  # Output: 6 (0110 in binary)
```
```
6
```

In [56]:
```python
# 4. ~ (Bitwise NOT): Inverts all the bits of the number. It changes 1 to 0 and 0 to 1.
# For signed integers, it also changes the sign (i.e., it calculates the two's complement).
a = 5  # 0101 in binary
result = ~a  # Bitwise NOT
print(result)  # Output: -6 (in two’s complement representation)
```
```
-6
```

In [57]:
```python
# 5. << (Left Shift): Shifts bits to the left by n positions, filling with zeros.
a = 5  # 00000101 in binary
result = a << 1  # Left shift by 1 bit
print(result)  # Output: 10 (00001010 in binary)
```
```
10
```

In [58]:
```python
# 6. >> (Right Shift): Shifts bits to the right by n positions, discarding bits and filling with sign bit (for negatives) or zeros (for positives).
a = 5  # 00000101 in binary
result = a >> 1  # Right shift by 1 bit
print(result)  # Output: 2 (00000010 in binary)
```
```
2
```

6. Membership Operators

Membership operators are used to check whether a value exists in a sequence

In [59]:
```python
# 1. in (Returns True if the value is present in the sequence)
text = "Hello, World!"
print("Hello" in text)
print("Python" in text)
```
```
True
False
```

In [60]:
```python
# 2. not in (Returns True if the value is NOT present in the sequence)
# Checking in a string
text = "Hello, World!"
print("Python" not in text)
print("World" not in text)
```
```
True
False
```

Identity Operators

Identity operators are used to compare the memory locations of two objects.

In [61]:
```python
# 1. is (Returns True if both variables refer to the same object in memory)
# Using integers
a = 10
b = 10
print(a is b)


x = "hello"
y = "hello"
print(x is y)

p = 3.14
q = 3.14
print(p is q)
```
```
True
True
False
```

In [62]:
```python
# 2. is not (Returns True if both variables do NOT refer to the same object in memory)
a = 10
b = 10
print(a is not b)

x = "hello"
y = "hello!"
print(x is not y)

p = 3.14
q = 3.14
print(p is not q)
```
```
False
True
True
```

Comments in Python

Comments are used to explain code and make it more readable. Python provides two types of comments:

Single-Line Comment (#): Use # at the beginning of a line to write a comment.

In [63]:
```python
# This is single line comment.
Text = "Hello"
print(Text)
```
```
Hello
```

Why use comments?

- To explain complex code
- To improve code readability

Control Statements in Python

Control statements help make decisions in Python based on conditions.

In [64]:
```python
# 1. if Statement: Executes a block of code only if the condition is True.
age = 18
if age >= 18:
    print("You are an adult.")
```
```
You are an adult.
```

In [65]:
```python
# 2. if-else Statement: Executes one block if the condition is True, otherwise executes the else block.
number = 10
if number % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```
```
Even number
```

In [66]:
```python
# 3. if-elif Statement: Used for multiple conditions. If one if or elif condition is True, it executes that block and skips the rest.
marks = 75
if marks >= 90:
    print("Grade: A")
elif marks >= 75:
    print("Grade: B")
elif marks >= 60:
    print("Grade: C")
else:
    print("Grade: F")
```
```
Grade: B
```

In [67]:
```python
# 4. if-elif-else Statement: Includes a final else to catch all remaining cases.
temperature = 15
if temperature > 30:
    print("It's hot outside.")
elif temperature > 20:
    print("It's warm.")
elif temperature > 10:
    print("It's cool.")
else:
    print("It's cold.")
```
```
It's cool.
```

In [68]:
```python
# 5. Nested if Statement: If inside another if for multiple conditions.
num = 10
if num > 0:
    if num % 2 == 0:
        print("Positive Even Number")
    else:
        print("Positive Odd Number")
```
```
Positive Even Number
```

In [69]:
```python
# 6. if with Logical Operators (and, or, not): Combining multiple conditions
x = 5
y = 10
if x > 0 and y > 0:
    print("Both numbers are positive.")
```
```
Both numbers are positive.
```

In [ ]:

In [ ]: