Python Output

In [1]:
```
# Python is case sensitive
print('Hello World')
```

```
Hello World
```

In [50]:
```
print(hey) # it throws error beacuse only string is always in " "
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9556\1311815035.py in <module>
----> 1 print(hey) # it throws error beacuse only string always in " "

NameError: name 'hey' is not defined
```

In [3]:
```
print(7)
```

```
7
```

In [4]:
```
print(True)
```

```
True
```

In [5]:
```
# how print function strng in python -- it print all the values which we want to pr
print('Hello',3,4,5,True)
```

```
Hello 3 4 5 True
```

In [6]:
```
# sep
print('Hello',3,4,5,True,sep='/') # sep is separator where in print function space
```

```
Hello/3/4/5/True
```

In [7]:
```
print('hello')
print('world')
```

```
hello
world
```

In [8]:
```
# end
print('you',end="=")
print ('me')
```

```
you=me
```

Data Types

1. Integers

In [9]:
```
print(8)
```

```
8
```

2. Float (Decimal)

In [10]:
```
print(8.44)
```

```
8.44
```

3. Boolean

In [11]:
```
print(True)
print(False)
```

```
True
False
```

4. String (Text)

In [12]:
```
print('Hello gem')
```

```
Hello gem
```

5. Complex

In [13]:
```
print(5+6j)
```

```
(5+6j)
```

6. List

In Python, a list is a data type used to store a collection of values. It is one of the built-in data types and is classified as a sequence type. Lists are ordered, mutable (which means you can change their contents), and can contain elements of different data types, including integers, floats, strings, or even other lists.

You can create a list in Python by enclosing a comma-separated sequence of values within square brackets [ ]. For example:

In [ ]:
```
print([1,2,3,4])
```

7. Tuple

In Python, a tuple is another data type used to store a collection of values, similar to a list. However, there are some key differences between tuples and lists:

- Immutable: The most significant difference is that tuples are immutable, meaning once you create a tuple, you cannot change its contents (add, remove, or modify elements). Lists, on the other hand, are mutable, and you can modify them after creation.
- Syntax: Tuples are created by enclosing a comma-separated sequence of values within parentheses (). Lists are created with square brackets []. For example:
- Performance: Due to their immutability, tuples can be more efficient in terms of memory and performance for certain use cases compared to lists.

Tuples are often used when you have a collection of values that should not be changed during the course of your program. For example, you might use tuples to represent

coordinates (x, y), dates (year, month, day), or other data where the individual components should remain constant.

In [14]:
```
print((1,2,3,4))
```

```
(1, 2, 3, 4)
```

8. sets

In Python, a set is a built-in data type used to store an unordered collection of unique elements. Sets are defined by enclosing a comma-separated sequence of values within curly braces {} or by using the built-in set() constructor. Sets automatically eliminate duplicate values, ensuring that each element is unique within the set.

In [15]:
```
print({1,2,3,4,5})
```

```
{1, 2, 3, 4, 5}
```

9. Dictionary

In Python, a dictionary is a built-in data type used to store a collection of key-value pairs. Each key in a dictionary maps to a specific value, creating a relationship between them. Dictionaries are also known as associative arrays or hash maps in other programming languages.

In [16]:
```
print({'name':'Nitish','gender':'Male','weight':70})
```

```
{'name': 'Nitish', 'gender': 'Male', 'weight': 70}
```

How to know which type of Datatype is?

In [18]:
```
type([1,2,3,4])
```

Out[18]:
```
list
```

In [19]:
```
type({'age':20})
```

Out[19]:
```
dict
```

3. Variables

In Python, variables are used to store data values. These values can be numbers, strings, lists, dictionaries, or any other data type. Variables are essential for manipulating and working with data in your programs. Here's how you declare and use variables in Python:

1. Variable Declaration: You declare a variable by assigning a value to it using the assignment operator '='.

In [21]:
```
x = 5  # Assigning the integer value 5 to the variable 'x'
name = "Alice"  # Assigning a string value to the variable 'name'
```

1. Variable Names: Variable names (also known as identifiers) must adhere to the following rules:

- They can contain letters (a-z, A-Z), digits (0-9), and underscores (_).
- They cannot start with a digit.
- Variable names are case-sensitive, so myVar and myvar are treated as different variables.
- Python has reserved keywords (e.g., if, for, while, print) that cannot be used as variable names.

1. Data Types: Python is dynamically typed, which means you don't need to declare the data type of a variable explicitly. Python will determine the data type automatically based on the assigned value.

In [22]:
```
x = 5  # 'x' is an integer variable
name = "Alice"  # 'name' is a string variable
```

1. Reassignment: You can change the value of a variable by assigning it a new value.

In [23]:
```
x = 5
x = x + 1  # Updating the value of 'x' to 6
```

In [24]:
```
print(x)
```

```
6
```

Multiple Assignment: Python allows you to assign multiple variables in a single line.

In [26]:
```
a, b, c = 1, 2, 3  # Assigning values 1, 2, and 3 to variables a, b, and c, respec
```

In [28]:
```
print(a)
```

```
1
```

In [29]:
```
a = 1
b = 2
c = 3
print(a,b,c)
```

```
1 2 3
```

In [30]:
```
a=b=c= 5
print(a,b,c)
```

```
5 5 5
```

What is comments in Python?

In Python, comments are used to annotate and provide explanations within your code. Comments are not executed by the Python interpreter; instead, they are meant for human readers to understand the code better. Comments are ignored by the interpreter during program execution.

Python supports two types of comments:

1. **Single-line Comments**: Single-line comments start with the hash symbol (`#`) and continue until the end of the line. They are used to add comments on a single line.

```
# This is a single-line comment
x = 5  # Assigning a value to 'x'
```

Everything after the `#` symbol on that line is considered a comment.

In [32]:
```
# This is a single-line comment
x = 5  # Assigning a value to 'x'
```

1. **Multi-line or Block Comments**: Python does not have a specific syntax for multi-line comments like some other languages (e.g., C, Java). However, you can create multi-line comments by using triple-quotes (`'''` or `"""`) as a string delimiter. These strings are not assigned to any variable and are ignored by the interpreter. This is a common practice for writing docstrings (documentation within functions and modules).

In [34]:
```
'''
This is a multi-line comment.
It can span multiple lines.
'''
def my_function():
    """
    This is a docstring.
    It provides documentation for the function.
    """
    pass
```

While these triple-quoted strings are not technically comments, they are often used as a way to document code effectively.

Comments are crucial for making your code more understandable, both for yourself and for others who may read your code. They help explain the purpose of variables, functions, and complex algorithms, making it easier to maintain and debug code. Good commenting practices can significantly improve code readability and maintainability.

4. User Input

How to get Input from the user in python?

In [35]:
```
input('Enter Email')
```

```
Enter EmailSagar@95
```

Out[35]:
```
'Sagar@95'
```

In [36]:
```
# take input from users and store them in a variable
fnum = int(input('enter first number'))
snum = int(input('enter second number'))
#print(type(fnum),type(snum))
# add the 2 variables
result = fnum + snum
# print the result
```

```
print(result)
print(type(fnum))
```

```
enter first number1
enter second number2
3
<class 'int'>
```

5. Type Conversion

How to covert One Datatypeinto another In python?

- int(): Converts a value to an integer data type. This is useful when you want to convert a string or a floating-point number to an integer.

In [44]:
```
str_number = "123"
int_number = int(str_number)  # Converts the string "123" to an integer
int_number
```

Out[44]:
```
123
```

- float(): Converts a value to a floating-point data type. It is used to convert integers or strings containing numeric values to floating-point numbers.

In [45]:
```
int_value = 42
float_value = float(int_value)  # Converts the integer 42 to a float
float_value
```

Out[45]:
```
42.0
```

In [37]:
```
a = 23
b = "24"

print(a+b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9556\3112067906.py in <module>
      2 b = "24"
      3
----> 4 print(a+b)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

It Gives error beacause We do not add float into integer datatyupe so we have to convert above opeartion

In [38]:
```
a = 23
b = "24"

print(float(a)+float(b))
```

```
47.0
```

6. Literals

In Python, literals are used to represent fixed values in your code. These values are not variables or expressions but rather constants that have a specific value and data type associated with them. Python supports various types of literals,

In [46]:
```
a = 0b1010 #Binary Literals
b = 100 #Decimal Literal
c = 0o310 #Octal Literal
d = 0x12c #Hexadecimal Literal

#Float Literal
float_1 = 10.5
float_2 = 1.5e2 # 1.5 * 10^2
float_3 = 1.5e-3 # 1.5 * 10^-3

#Complex Literal
x = 3.14j

print(a, b, c, d)
print(float_1, float_2,float_3)
print(x, x.imag, x.real)
```

```
10 100 200 300
10.5 150.0 0.0015
3.14j 3.14 0.0
```

In [48]:
```
# binary
x = 3.14j
print(x.imag)
```

```
3.14
```

In [49]:
```
string = 'This is Python'
strings = "This is Python"
char = "C"
multiline_str = """This is a multiline string with more than one line code."""
unicode = u"\U0001f600\U0001F606\U0001F923"
raw_str = r"raw \n string"

print(string)
print(strings)
print(char)
print(multiline_str)
print(unicode)
print(raw_str)
```

```
This is Python
This is Python
C
This is a multiline string with more than one line code.
😀😆🤣
raw \n string
```