Python : From Zero to All of Your Basic Needs in Python

Name: **Sheikh Muhammad Ashik**
University Of Asia Pacific
Department of Computer Science and Engineering
Email: dev.smashik@gmail.com
Linkedin: https://www.linkedin.com/in/smashik/

All about Python Programming

Python is a high-level, interpreted, and general-purpose programming language. It was created by Guido van Rossum and first released in 1991. Python emphasizes readability and simplicity, and it has become one of the most popular programming languages due to its versatility and ease of use.

Here are some key characteristics and aspects of Python programming:

Readability:

Python is designed to be easily readable, with a clean and concise syntax. This readability makes it easier to write and maintain code.

Interpreted Language:

Python is an interpreted language, meaning that the source code is executed line by line by an interpreter. This allows for dynamic typing and makes the development process more interactive.

High-Level Language:

Python is a high-level language, which means that it abstracts away low-level details like memory management and provides a more user-friendly environment.

Versatility:

Python is a versatile language that can be used for various purposes, including web development, data analysis, artificial intelligence, machine learning, automation, scripting, and more.

Object-Oriented Programming (OOP):

Python supports object-oriented programming principles, allowing developers to create and use classes and objects for code organization and reusability.

Large Standard Library:

Python comes with a large standard library that provides modules and packages for a wide range of tasks. This reduces the need for developers to write code from scratch for common functionalities.

Community and Ecosystem:

Python has a vibrant and active community of developers. There is a vast ecosystem of third-party libraries and frameworks that enhance Python's capabilities in various domains. Cross-Platform Compatibility:

Python is a cross-platform language, meaning that Python code can run on different operating systems without modification.

Open Source:
Python is an open-source language, which means that its source code is freely available, and the Python community can contribute to its development and improvement.

Dynamic Typing:

Python is dynamically typed, which means that the type of a variable is interpreted during runtime. This provides flexibility but also requires careful attention to variable types. Python's popularity has grown significantly over the years, and it is widely used in various industries and fields. Its simplicity, readability, and extensive community support make it an excellent choice for beginners and experienced developers alike.

Basics of Python

Print() Function

In Python, the print() function is used to output text or other data to the console. It's a built-in function, and it's commonly used for debugging, displaying information to the user, or just printing the results of a program.

Here are some details about print() Functions:

In [2]:
```
#Print About My Self
print("Hello, I'm Sheikh Muhammad Ashik and I'm UI/UX Designer, Blogger, Programmer, Hacker, W
```

```
Hello, I'm Sheikh Muhammad Ashik and I'm UI/UX Designer, Blogger, Programmer, Hacker, Web Des
igner and CTF (Capture The Flag Player)
```

In [3]:
```
#Printing Multiple Values
Name = "Sheikh Muhammad Ashik"
Age = 24
Phone = "+8801621942031"
University = "University of Asia Pacific"
Semester = "3rd year 1st semester"
Department = "Computer Science and Engineering"
Skills = "UI/UX, Programming, Blogging, Hacking, Web Designing"
Programming_Language = "Pyhton"

print(f"Hi I'm {Name}, my age is {Age}, my phone number is {Phone}, I'm student of the departm
```

```
Hi I'm Sheikh Muhammad Ashik, my age is 24, my phone number is +8801621942031, I'm student of
the department of Computer Science and Engineering at University of Asia Pacific in 3rd year
1st semester, my core skills are UI/UX, Programming, Blogging, Hacking, Web Designing and las
tly my favorite programming language is Pyhton
```

In [8]:
```
# Fomate Spicifiers
pi = 3.141596745453454364567
print("Value of PI: {:.2f}". format(pi)) #This {:.2f} will print the value of pi with two deci
```

```
Value of PI: 3.14
```

In [12]:
```
# Changing Seperator and end characters
print("Sheikh", "Muhammad", "Ashik", "21201118@uap-bd.edu", "+8801621942031", sep=',,, ', end=

#sep=',,' - for each
```

```
Sheikh,,, Muhammad,,, Ashik,,, 21201118@uap-bd.edu,,, +8801621942031 .... Etc
```

Variable assignment

variable assignment is the process of associating a name (variable) with a value.
Here are some details about variable assignment in Python:

Dynamic Typing

Python is a dynamically typed language, meaning you don't have to explicitly declare the type of a variable. The type of a variable is determined at runtime based on the assigned value.

In [19]:
```
x = 100  # Integer
y = 3.1416 # Float
name = "Ashik" # String

print(x,y, name)
```

```
100 3.1416 Ashik
```

In [18]:
```
#Multiple Assignment

a, b, c = 1, 2, 3
print(a,b,c)
```

```
1 2 3
```

Swapping Values

Python allows you to easily swap values between variables without using a temporary variable:

In [21]:
```
a = 10
b = 70

a, b = b, a

print(a, b)
```

```
70 10
```

Constants

While Python doesn't have a specific constant keyword, it's a convention to use uppercase names for constants:

In [22]:
```
PI = 3.1416
Gravity = 9.8

print(PI, Gravity)
```

```
3.1416 9.8
```

Python operations

various types of operations can be performed, including arithmetic, comparison, logical, bitwise, and more. Here are details about some common types of operations:

In [27]:
```
#Arithmetic Operations

a = 50
b = 20

addition = a + b
```

```
subtraction = a - b
multiplication = a * b
division = a / b
floor_division = a // b
Modulus = a % b
Exponentiation = a**b

print(addition, subtraction, multiplication, division, floor_division, Modulus, Exponentiation
```

```
70 30 1000 2.5 2 10 95367431640625000000000000000000000
```

In [28]:
```
# Comparison Operations

x = 50
y = 70

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

print(equal, not_equal, greater_than, less_than, greater_than_or_equal_to, less_than_or_equal_
```

```
False True False True False True
```

In [37]:
```
# Logical Operations

p = True
q = False

logical_and = p and q
logical_or = p or q
logical_not = not p

print(logical_and, logical_or,logical_not)
```

```
False True False
```

In [38]:
```
# Bitwise Operations

a = 5 # binary 101
b = 3 # binary 011

bitwise_and = a & b
bitwise_or = a | b
bitwise_not = ~a
bitwise_xor = a ^ b
left_shift = a << 1
right_shift = a >> 1

print(bitwise_and, bitwise_or, bitwise_not, bitwise_xor, left_shift, right_shift)
```

```
1 7 -6 6 10 2
```

& - Bitwise and

| - Bitwise or

~ - Bitwise not

^ - Bitwise Xor

<< - Bitwise Left Shift

'>>' - Bitwise Right Shift

Precedence: ~(Highest), << >>, &, ^, |(lowest)

In [43]:
```
x = 5
y = 4
z = 3

# Precedence: ~(Highest), << >>, &, ^, |(Lowest)
```

```
ans = x & y | z >> 1
print(ans)
```

```
5
```

Python Strings

a string is a sequence of characters. Strings are one of the fundamental data types, and they are used to represent and manipulate text. Strings in Python are immutable, meaning their values cannot be changed after they are created. Here are some key aspects of strings in Python:

In [51]:
```
# Creating String
# You can create strings using single (' '), double (" "), or triple (''' ''' or """ """) quot

single_quoted = 'Hello, Python Developer!'
double_quoted = "are you enjoy learnig Python?"
triple_quoted = '''You will be surprised to know you can
use string in multiple line in triple quoted string
'''

print(single_quoted)
print(double_quoted)
print(triple_quoted)
```

```
Hello, Python Developer!
are you enjoy learnig Python?
You will be surprised to know you can
use string in multiple line in triple quoted string
```

In [52]:
```
# Accessing Characters

name = "Sheikh Muhamad Ashik"
first_character = name[0]
nineth_character = name[8]

print(first_character, nineth_character)
```

```
S u
```

In [56]:
```
# String Slicing

name = "Sheikh Muhammad Ashik"
substring = name[0:9] # start point 0: end point 9
print(substring)
```

```
Sheikh Mu
```

In [58]:
```
# String Concatenation

name1 = "Sheikh"
name2 = "Muhammad"
name3 = "Ashik"

concatenation = name1+ "_" + name2+ "_" + name3
print(concatenation)
```

```
Sheikh_Muhammad_Ashik
```

In [60]:
```
# String Repetition
name = "Sheikh Muhammad Ashik "
repetition = name * 4 # 4 times repetition
print(repetition)
```

```
Sheikh Muhammad Ashik Sheikh Muhammad Ashik Sheikh Muhammad Ashik Sheikh Muhammad Ashik
```

In [63]:
```
 #String Methods
```

String Method

Python provides a variety of built-in methods for working with strings. Some common methods include:

len(): Returns the length of the string.
lower(), upper(): Converts the string to lowercase or uppercase.
strip(): Removes leading and trailing whitespaces.
replace(old, new): Replaces occurrences of a substring with another substring.
split(delimiter): Splits the string into a list of substrings based on a delimiter.

In [74]:
```
string = "Python is awesome!"

length = len(string)

uppercase_string = string.upper()
lowercase_string = string.lower()
strip_string = string.strip()
replace_string = string.replace("Python is awesome!", "Python is populer programming Language"
split_string = string.split()

print("total character is: ", length)
print(lowercase_string)
print(uppercase_string)
print(strip_string)
print(split_string)
print(replace_string)
```

```
total character is:  18
python is awesome!
PYTHON IS AWESOME!
Python is awesome!
['Python', 'is', 'awesome!']
Python is populer programming Language
```

String Formatting

Python supports different ways of formatting strings, including the % operator and the format() method. Additionally, f-strings (formatted string literals) were introduced in Python 3.6:

In [81]:
```
#String Formatting

name = "Sheikh Muhammad Ashik"
age = 24
department = "Computer Science and Engineering"

# f_string = Formated String
f_string1 = "My name is %s and I am %d years old" %(name, age)
f_string2 = "My name is {} and I am {} years old".format(name, age)
f_string3 = f"My name is {name} and I am {age} years old"
print(f_string1)
print(f_string2)
print(f_string3)
```

```
My name is Sheikh Muhammad Ashik and I am 24 years old
My name is Sheikh Muhammad Ashik and I am 24 years old
My name is Sheikh Muhammad Ashik and I am 24 years old
```

Escape Characters

Special characters in strings are often represented using escape characters, such as \n for a newline or \t for a tab:

In [82]:
```
new_line = "Sheikh \nMuhammad\t Ashik"
print(new_line)
```

```
Sheikh
Muhammad	 Ashik
```

String Immutability

Strings in Python are immutable, meaning you cannot modify their individual characters. However, you can create new strings based on existing ones.

In [85]:
```
string = "Ashik"
Immutable_string = "N" + string[1:]
print(Immutable_string)
```

```
Nshik
```

Conrtrol-Flow

Control flow in Python refers to the order in which statements are executed based on certain conditions or loops. Python provides several control flow structures to help you control the flow of your program. Here are the main control flow structures in Python:

Conditional Statements (if, elif, else): 1. The if statement is used to check a condition, and if it's true, the indented block of code following it is executed.
2. elif (else if) allows you to check multiple conditions if the previous ones are not true.
3. else is used to define a block of code to be executed if none of the conditions in the if or elif statements are true.

In [1]:
```
# if, elif, else

x = 50
if x > 0:
    print("x is positive")
elif(x == 0):
    print("x  is zero")
else:
    print("x is negative")
```

```
x is positive
```

Loops (for and while):

for loops iterate over a sequence (such as a list, tuple, or string) and execute a block of code for each item in the sequence. while loops continue to execute a block of code as long as a specified condition is true.

In [4]:
```
# For Loop

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

```
0
1
2
3
4
```

In [ ]:
```
# While

count = 0
while count < 5:
    print(count)
    count += 1
```

Break and Continue Statements:

break is used to exit a loop prematurely. continue is used to skip the rest of the code inside a loop for the current iteration and move to the next iteration.

In [10]:
```
for i in range(15):
    if i == 5:
        break
    print(i)
print()

for i in range(15):
    if i == 5:
        continue
    print(i)
```

```
0
1
2
3
4

0
1
2
3
4
6
7
8
9
10
11
12
13
14
```

Exception Handling (try, except, finally):

try is used to wrap a block of code that might raise an exception. except is used to handle specific exceptions that might occur within the try block. finally is used to define a block of code that will be executed no matter what, whether an exception occurred or not.

In [11]:
```
try:
    result = 10/0
except ZeroDivisionError:
    print("Cannot divide by zero")
finally:
    print("This will executed regardless of an exception")
```

```
Cannot divide by zero
This will executed regardless of an exception
```

User Input

In Python, you can use the input() function to read user input from the console. The input() function takes a string as an argument (the prompt message), displays it to the user, and then

waits for the user to enter some text. Here's a simple example:

In [15]:
```
# User Input
name = input("Enter Your name: ")  # String Input
age = int(input("Enter your age: ")) # Integer Input

print(name, str(age))
```

```
Enter Your name: Ashik
Enter your age: 24
Ashik 24
```

Comments

In [6]:
```
# here is a single line comment

'''
here is
a multiple
line comment
'''

print("hello")
```

```
hello
```

Loops

Loops in Python are control flow structures that allow you to execute a block of code repeatedly. Python provides two main types of loops: for and while. Let's explore each of them:

In [7]:
```
#for variable in sequence:
    # Code to be executed for each item in the sequence
```

In [ ]:
```
names = ["Sheikh", "Muhammad", "Ashik"]
for i in names:
    print(i)
```

In [1]:
```
# While Loop

count = 0
while(count < 5):
    print(count)
    count += 1
```

```
0
1
2
3
4
```

In [2]:
```
# Using break to exit the loop when reaching 3
for i in range(5):
    if i == 3:
        break
    print(i)
```

```
0
1
2
```

In [3]:
```
# Using continue to skip printing 3
for i in range(5):
    if i == 3:
        continue
    print(i)
```

```
0
1
2
4
```

List in Python

a list is a built-in data type used to store an ordered collection of items. Lists are versatile and can hold elements of different data types, including numbers, strings, and even other lists. Lists are mutable, meaning you can modify their contents by adding or removing elements. Here's a basic overview of lists in Python:

In [17]:
```python
# Creating List in python

list = [1, 2, 3, 4, 5, 'Sheikh', 'Muhammad', 'Ashik']
print(list)
```

```
[1, 2, 3, 4, 5, 'Sheikh', 'Muhammad', 'Ashik']
```

In [18]:
```python
# Accessing Elements
print(list[0]) # Output: 1
print(list[7]) # Output: Ashik
print(list[-2]) # Output: Muhammad
```

```
1
Ashik
Muhammad
```

In [19]:
```python
# Slicing Lists

print(list[1:4]) # Output: [2, 3, '4'] (from index 1 to 3, excluding 4)
```

```
[2, 3, 4]
```

In [20]:
```python
# Modifying List

list[2] = "Python"
list.append(4)
list.insert(2, 'Programming')
list.remove(5)

print(list)
```

```
[1, 2, 'Programming', 'Python', 4, 'Sheikh', 'Muhammad', 'Ashik', 4]
```

List Operation

Lists support various operations, such as concatenation (+), repetition (*), and checking membership (in).

In [23]:
```python
list1 = [1, 5, 7]
list2 = ["Sheikh", "Muhammad", "Ashik"]

connect_list = list1 + list2
repeated_list = list1 * 2
check_membership = 'Python' in list2

print(connect_list)
print(repeated_list)
print(check_membership)
```

```
[1, 5, 7, 'Sheikh', 'Muhammad', 'Ashik']
[1, 5, 7, 1, 5, 7]
False
```

List Methods

Lists have built-in methods for various operations, such as sorting, reversing, finding the length, and more.

In [25]:
```
numbers = [9, 5, 6, 2, 1, 3, 0, 4]
n1 = numbers.sort()
n2 = numbers.reverse()
length = len(numbers)

print(n1)
print(n2)
print(length)
```

```
None
None
8
```

List Comprehensions:

List comprehensions provide a concise way to create lists.

In [26]:
```
squares = [x**2 for x in range(5)]
print(squares)
```

```
[0, 1, 4, 9, 16]
```

Tuples

In Python, a tuple is a built-in data type similar to a list, but with a few key differences. A tuple is an ordered, immutable collection of elements. Once a tuple is created, its elements cannot be modified or changed. Tuples are created by enclosing a sequence of elements within parentheses () and separating them with commas. Here's an overview of tuples in Python:

You can create a tuple by enclosing a sequence of elements within parentheses ().

In [35]:
```
tuple1 = (1, 2, "Ashik", "Python")
print(tuple)
```

```
(1, 2, 'Ashik', 'Python')
```

In [36]:
```
# Accessing Tuple

print(tuple1[0])  # Output: 1
print(tuple1[2])  # Output: 'Ashik'
print(tuple1[-1]) # Output: Python (negative index refers to elements from the end)
```

```
1
Ashik
Python
```

In [37]:
```
# Slicing Tuples

print(tuple1[1:3]) # print index 1 to 2
```

```
(2, 'Ashik')
```

Immutable Nature:

Unlike lists, tuples are immutable. Once a tuple is created, you cannot change, add, or remove elements from it.

In [38]:
```
tuple1[2] = "Sheikh"
print(tuple)
```

```
# TypeError: 'tuple' object does not support item assignment
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 tuple1[2] = "Sheikh"
      2 print(tuple)

TypeError: 'tuple' object does not support item assignment
```

Common Use Cases: Tuples are often used to represent records or collections of related data. They are used in functions that return multiple values. Tuples can be used as keys in dictionaries (since they are immutable).

When to Use Tuples: Use tuples when you have a collection of items that should remain constant throughout the program. Use tuples for heterogeneous data (mix of different data types). Use tuples for read-only data.

Dictionary in Python

dictionary is a built-in data type that represents a collection of key-value pairs. It is an unordered and mutable collection, meaning that the elements can be modified, and there is no specific order maintained among the elements. Dictionaries are implemented as hash tables, making them efficient for lookups and retrievals. Here's an overview of dictionaries in Python:

Creating a Dictionary: A dictionary is created by placing a comma-separated list of key-value pairs within curly braces {}.

In [57]:
```
# Create Dictionary
dictionary = {'name' : "Ashik", 'age' : 24, 'city' : "Dhaka"}
print(dictionary)
```

```
{'name': 'Ashik', 'age': 24, 'city': 'Dhaka'}
```

In [58]:
```
# Accessing Values

print(dictionary['name'])
print(dictionary['age'])
```

```
Ashik
24
```

In [59]:
```
# Modifying dictionary

dictionary['age'] = 45
dictionary['occupation'] = "Senior Software Engineer"

print(dictionary)
```

```
{'name': 'Ashik', 'age': 45, 'city': 'Dhaka', 'occupation': 'Senior Software Engineer'}
```

In [60]:
```
# Removing Item from dictionary

del dictionary['city'] #remove items from a dictionary using the del keyword.
print(dictionary)
```

```
{'name': 'Ashik', 'age': 45, 'occupation': 'Senior Software Engineer'}
```

Dictionary Methods:

Dictionaries have built-in methods for various operations, such as getting keys, values, items, and more.

In [61]:
```
keys = dictionary.keys()
values = dictionary.values()
items = dictionary.items()

print(keys)
print(values)
print(items)
```

```
dict_keys(['name', 'age', 'occupation'])
dict_values(['Ashik', 45, 'Senior Software Engineer'])
dict_items([('name', 'Ashik'), ('age', 45), ('occupation', 'Senior Software Engineer')])
```

In [62]:
```
# Dictionary Comprehensions

# Like lists, dictionaries also support comprehensions for concise creation.

square = {x: x**2 for x in range(5)}
print(square)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

In [68]:
```
# Nested Dictonary
# Dictionaries can contain other dictionaries as values, creating nested structures.

nested_dictionary ={'person' : { 'name' : 'Ashik', 'age' : 24}}
print(nested_dictionary)
```

```
{'person': {'name': 'Ashik', 'age': 24}}
```

Python Function

a function is a block of organized, reusable code that performs a specific task. Functions provide modularity and help in code reusability. They allow you to break down a program into smaller, manageable pieces, making it easier to understand, debug, and maintain. Here's an overview of functions in Python:

Defining a Function:

In Python, you can define a function using the def keyword, followed by the function name, a pair of parentheses (), and a colon :. The code block that follows the colon is the body of the function.

In [71]:
```
# Syntax

def function_name(parameters):
    # Function Body
    # Code to perform a specific task
    return result # optional: return statement
```

In [75]:
```
def greet(name):
    print("Hello, " + name + "!")

greet("Ashik")
```

```
Hello, Ashik!
```

Parameters and Arguments:

Parameters are variables that are defined in the function signature and act as placeholders for the values that will be passed to the function. Arguments are the actual values passed to the function when it is called.

In [79]:
```
def add_num(x, y):
    total = x + y
    return total
```

```
result = add_num(5,3)
print(result)
```

```
8
```

Return Statement:

A function can optionally return a value using the return statement. The returned value can be assigned to a variable or used directly in the calling code.

In [81]:
```
def square(x):
    return x**2
result = square(3)
print(result)
```

```
9
```

Default Parameters:

You can provide default values for parameters in a function. If a value is not provided when calling the function, the default value is used.

In [83]:
```
def power(base, exponent = 2):
    return base ** exponent

result1 = power(2)
result2 = power(4)

print(result1)
print(result2)
```

```
4
16
```

Variable Number of Arguments:

You can define functions that accept a variable number of arguments using *args (for positional arguments) and **kwargs (for keyword arguments).

In [84]:
```
def print_arguments(*args, **kwargs):
    print("Positional arguments: ", args)
    print("Keyword arguments: ", kwargs)

print_arguments(1, 2, 3, name="Ashik", age=24)
```

```
Positional arguments:  (1, 2, 3)
Keyword arguments:  {'name': 'Ashik', 'age': 24}
```