Assignment: Python Basics Part 1

Instructions:

1. Open this assignment in Google Colab.
2. Follow the guided instructions in each exercise.
3. Complete the tasks by filling in the blanks or modifying the code where prompted.
4. Run the code cells to see the output and understand the logic.
5. Run your cells using SHIFT+ENTER (or "Run cell")
6. Write code in the designated areas.
7. Do not modify the code outside of the designated areas

After this assignment you will:

```
Be able to use iPython Notebooks.
Be able to code & understand basic python.
Build a basic foundation for future concepts.
```

Let's get started!

Variable Declarations & Printing in Python

Here we will understand how to declare variable in python and print them.

- **Task**: Declare three variables and understand their data types.
- **Steps**:
    1. Assign the following values:
        - `x` as an integer (e.g., 10)
        - `y` as a float (e.g., 20.5)
        - `z` as a string (e.g., "Python")
    2. Print each variable.
    3. Print the type of each variable using `type()`.

```
# Step 1: Declare variables
x = 5  # Fill in an integer value
y = 10.1  # Fill in a float value
z = "mic"  # Fill in a string value

# Step 2: Print variables
print("x:", x)
print("y:", y)
print("z:", z)

# Step 3: Print data types
print("Type of x:", type(x))
print("Type of y:", type(y))
print("Type of z:", type(z))

# Step 4: Try to declare a new variable and print its value.
# Start your code here #
```

```
x: 5
y: 10.1
z: mic
Type of x: <class 'int'>
Type of y: <class 'float'>
Type of z: <class 'str'>
```

Dynamic Printing

Trying to understand how to take a variable directly from the user and print its value.

- **Task**: Declare a variable and print it dynamically by taking user's input.
- **Steps**:
    1. Assign a variable and take user's input as its value.
    2. Print the variable.

```
#Step 1: Take user's input.
user_num = int(input("Please enter your number : "))

#Step 2: Print the number.
# Making use of f-string literal, which is used to add values within curly braces.
print(f"Your number is {user_num}")


#Step 2: Take user's string input.
user_num = str(input("Please enter your string : "))

#Step 2: Print the number.
# Making use of f-string literal, which is used to add values within curly braces.
print(f"Your number is {user_num}")
```

```
Please enter your number : 9
Your number is 9
Please enter your string : gaurav
Your number is gaurav
```

Working with basic data types.

There are different data types in python. In the previous questions, you worked with integers, float, complex and strings. Now, We will work and understand List, Tuple, Range & Dictionary.

```
The list can be represented by [ ], e.g. [1, 2, 3, 4, 5]
A tuple can be represented by  ( ), e.g. (1, 2, 3, 4, 5)
The set can be represented by { }, e.g. {1, 2, 3, 4, 5}
The dictionary can be represented by { }, e.g. {"name":"John", "age":45}
```

Creating these data-types

```
new_list = []
new_tuple = ()
new_set = set()
new_dict = {}
```

Task: We will be understanding how to work with list, tuple, range and dictionaries.

Steps:

```
1.)Create a list fruits with the values "apple", "banana", "cherry".
2.)Create a tuple coordinates with values 10, 20, 30.
3.)Create a dictionary student with keys "name", "age", and "grade" Assign values "Alice", 22, "A".
```

Print each data structure.

```
# Step 1: Define the data structures
fruits = [] # Fill in a list of strings, e.g., ["apple", "banana", "cherry"]
coordinates = ()  # Fill in a tuple of numbers, e.g., (10, 20, 30)
student = {"name": __, "age": __}  # Fill in values

# Step 2: Print the data structures
print("Fruits list:", fruits)
print("Coordinates tuple:", coordinates)
print("Student dictionary:", student)
```

```
Fruits list: []
Coordinates tuple: ()
Student dictionary: {'name': '', 'age': ''}
```

Working with Tuple

1. Create a tuple colors containing random numbers.
2. Access and print the second element of the tuple.
3. Try to modify the third element of the tuple and observe the result.

Gemini

```
# Step 1: Define the tuple
number_tuple = (5, 4, 3, 2, 1)

# Step 2: Access and print the second element
print("Second number:-", number_tuple[1]) # Try to print the second number from the tuple.
print("Second number from reverse order:-", number_tuple[-4]) # Try to print the second number from the tuple.

print("Third number:", number_tuple[2])  # Using indexing to access the third element from the tuple.

# Step 3: Try modifying the tuple (uncomment the line below and run it)
number_tuple[2] = 10
print(number_tuple)
#number_tuple[2] = 10
#print(number_tuple)

# Try to understand what happened after you tried to change a tuple.
# Tuples are immutable, meaning you cannot change, add, or remove items once the tuple is created. Attempting to do so will result
```

```
Second number:- 4
Second number from reverse order:- 4
Third number: 3
```

Working with Sets

1. Create a set prime_numbers with the values 2, 3, 5, 7, 11
2. Add 13 to the set using add().
3. Remove 7 from the set using remove().

Print the final Set.

```
# Step 1: Define the set
prime_numbers = {2, 3, 5, 7, 11} # Fill in a set, e.g., {2, 3, 5, 7, 11}

# Step 2: Modify the set
prime_numbers.add(13)  # Add 13
prime_numbers.remove(7)  # Remove 7

# Step 3: Print the final set
print("Final set of prime numbers:", prime_numbers)
```

```
Final set of prime numbers: {2, 3, 5, 11, 13}
```

Working with Dictionaries

1. Create a dictionary inventory with keys "apples", "bananas", and "oranges" having values 10, 20, and 15, respectively.
2. Add a new key-value pair: "grapes": 12.
3. Update the value of "bananas" to 25.
4. Remove the key "oranges" using pop().

Print the final dictionary and its length.

---

1. Start with the dictionary car = {"brand": "Toyota", "model": "Corolla", "year": 2020}.
2. Add a new key-value pair: "color": "red".
3. Update the value of year to 2022.
4. Remove the key model.

Print the final dictionary.

```
# Step 1: Define the dictionary
inventory = {"apples": 10, "bananas": 20, "oranges": 15}  # Fill in values, for example in apples.
```

```
# Step 2: Modify the dictionary
inventory["grapes"] = 12      # Add "grapes": 12
inventory["bananas"] = 25     # Update "bananas" to 25
inventory.pop("oranges")      # Remove "oranges"

# Step 3: Print the dictionary and its length
print("Updated inventory:", inventory)
print("Number of items in inventory:", len(inventory))
```

```
Updated inventory: {'apples': 10, 'bananas': 25, 'grapes': 12}
Number of items in inventory: 3
```

```
# Step 1: Define the dictionary
car = {"brand": "Toyota", "model": "Corolla", "year": 2020}

# Step 2: Modify the dictionary
car["color"] = "red"   # Add "color": "red"
car["year"] = 2022     # Update "year" to 2022
del car["model"]       # Remove the "model" key

# Step 3: Print the modified dictionary
print("Updated car dictionary:", car)
```

```
Updated car dictionary: {'brand': 'Toyota', 'year': 2022, 'color': 'red'}
```

Working with List

1. Start with the list numbers = [1, 2, 3, 4, 5].
2. Add 6 to the end of the list using append().
3. Insert 0 at the beginning of the list using insert().
4. Remove the number 3 using remove().
5. Sort the list in ascending order using sort().

Print the modified list and its length.

```
# Step 1: Define the list
numbers = [1, 2, 3, 4, 5]

# Step 2: Modify the list
numbers.append(6)  # Add 6 in the bracket.
numbers.insert(0, 0)  # Insert 0 at index 0: num.insert(number to be added, the index you want to add)
numbers.remove(2)  # Remove 3
numbers.sort()  # Sort the list

# Step 3: Print the modified list
print("Modified list:", numbers)
print("Length of the list:", len(numbers))
```

```
Modified list: [0, 1, 3, 4, 5, 6]
Length of the list: 6
```

Basic arithmetic operations

Understanding how we will perform basic mathematical operations in python.

Declare variables a = 8 and b = 3 in your code. Perform and print results for the following operations:

- Addition
- Subtraction
- Multiplication
- Division
- Modulus
- Exponentiation
- Floor division

Hint: Use operators like +, -, *, /, %, //, and **.

```
# Step 1: Declare variables
a = 9  # Fill in a value, e.g., 8
```

```
b = 4  # Fill in a value, e.g., 3

# Step 2: Perform operations
print("Addition (a + b):", a + b)
print("Subtraction (a - b):", a - b)
print("Multiplication (a * b):", a * b)
print("Division (a / b):", a / b)
print("Modulus (a % b):", a % b)
print("Exponentiation (a ** b):", a ** b)
print("Floor Division (a // b):", a // b)
```

```
Addition (a + b): 13
Subtraction (a - b): 5
Multiplication (a * b): 36
Division (a / b): 2.25
Modulus (a % b): 1
Exponentiation (a ** b): 6561
Floor Division (a // b): 2
```

Understanding Control Flow & Operators

Control flow and operators are essential in programming because they form the foundation of logic and decision-making, enabling developers to create dynamic, efficient, and functional programs.

Relational Operators

```
Used for comparing values. Returns a boolean value
```

```
# Step 1: Declare two numbers
a = 15
b = 10

# Step 2: Compare the numbers
print("Is a equal to b?", a == b)  # Equality check
print("Is a not equal to b?", a != b)  # Not equal check
print("Is a greater than b?", a > b)  # Greater than check
print("Is a less than or equal to b?", a <= b)  # Less than or equal check
```

```
Is a equal to b? False
Is a not equal to b? True
Is a greater than b? True
Is a less than or equal to b? False
```

Logical Operators

```
AND
OR
NOT
```

```
# Step 1: Input a number
num = int(input("Enter a number: "))

# Step 2: Use logical operators
if num > 0 and num % 2 == 0:
    print("The number is positive and even.")
elif num > 0 and num % 2 != 0:
    print("The number is positive and odd.")
else:
    print("The number is either zero or negative.")
```

```
Enter a number: 10
The number is positive and even.
```

Control Flow (If-Elif)

Control flow structures allow programs to make decisions based on conditions, enabling dynamic behavior. Example: Executing different actions based on user input or system states.

```
# Step 1: Input a number
num = int(input("Enter a number: "))

# Step 2: Check the conditions
if __:  # Replace __ with a condition to check if num > 0
    print("The number is positive.")
elif __:  # Replace __ with a condition to check if num < 0
    print("The number is negative.")
else:
    print("The number is zero.")
```

```
Enter a number: 17
The number is zero.
```

```
# Step 1: Input student marks
marks = int(input("Enter your marks (0-100): "))

# Step 2: Categorize the student
if marks >= 90:  # Replace __ with a condition to check marks >= 90
    print("Grade: A")
elif marks >= 75:  # Replace __ with a condition to check marks >= 75
    print("Grade: B")
elif marks >= 50:  # Replace __ with a condition to check marks >= 50
    print("Grade: C")
else:
    print("Grade: F")
```

```
Enter your marks (0-100): 75
Grade: B
```

Loops in Control Flow

Now, we will learn to execute a code through loops. There are two ways to utilise loops. One being the for loop and the other being the while loop.

The syntax for these are given below:

```
for i in range(0, 10):
  ##code to be executed.

while(true):
  ##code to be executed
  ##exiting the loop.
```

```
# Step 1: Input a number
num = int(input("Enter a number: "))

# Step 2: Use a loop to generate the table
print("Multiplication Table:")
for i in range(1, 11):
    print(num * i)  # Replace __ with the multiplication formula, e.g., num * i

# Step 3: Print only the even numbers from 1 to 50.
for num in range(51):
  if num % 2 == 0:
    print(num)
```

```
Enter a number: 12
Multiplication Table:
12
24
36
48
60
72
84
96
```

```
108
120
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
```

```
# Step 1: Input the value of n
n = int(input("Enter a positive integer: "))

# Step 2: Initialize variables
sum = 0
i = 1

# Step 3: Use a while loop
while i <= n:  # Replace __ with a condition to run while i <= n
    sum += i
    i += 1

print("Sum of numbers from 1 to", n, "is:", sum)
```

```
Enter a positive integer: 12
Sum of numbers from 1 to 12 is: 78
```

Use of Break and Continue

break

- Immediately exits the loop
- Stopping further iterations.
- It's useful for terminating a loop early when a specific condition is met, improving efficiency.

continue

- Skips the current iteration and moves to the next one.
- It's helpful for ignoring certain cases without stopping the entire loop, maintaining control over logic flow.

```
# Step 1: Use a for loop to iterate from 1 to 10
for i in range(1, 11):
    if i == 7:  # Condition to stop the loop
        print("Encountered 7, stopping the loop.")
        break  # Stop the loop
    print(i)  # Print the current number


# Execute a code where it prints numbers from 1 to 10 but when we encounter 5 it breaks.
```

```
1
2
3
4
5
6
Encountered 7, stopping the loop.
```

```
# Step 1: Use a for loop to iterate from 1 to 10
for i in range(1, 11):
    if i == 5:  # Condition to skip printing
        continue  # Skip this iteration
    print(i)  # Print the current number

# Code a loop which continues on even numbers and prints odd numbers in the range(0 to 10).
```

```
1
2
3
4
6
7
8
9
10
```