Various types of Basic Print Functions in Python

Question 1: Simple Print

Q: Write a Python program that prints "Hello, World!".

Solution:

```
print("Hello, World!")
```

Question 2: Print Variables

Q: Write a Python program that defines two variables, name and age, and prints them in the format: "Name: [name], Age: [age]".

Solution:

```
name = "Alice"

age = 30

print("Name:", name, "Age:", age)
```

Question 3: String Concatenation

Q: Write a Python program that prints "Python" and "Programming" on the same line using string concatenation.

Solution:

```
print("Python" + " " + "Programming")
```

Question 4: Formatting Strings

Q: Write a Python program that uses the format method to print "Alice is 30 years old".

```
name = "Alice"

age = 30
```

print("{} is {} years old".format(name, age))

Question 5: f-Strings

Q: Write a Python program that uses an f-string to print "Alice is 30 years old".

Solution:

```
name = "Alice"
age = 30
print(f"{name} is {age} years old")
```

Question 6: Printing Multiple Lines

Q: Write a Python program that prints the following:

```
Line 1
Line 2
Line 3
```

Solution:

```
print("Line 1")
print("Line 2")
print("Line 3")
```

Question 7: Escape Characters

Q: Write a Python program that prints the following:

```
She said, "Hello, World!"
```

Solution:

```
print('She said, "Hello, World!"')
```

Question 8: Using Newline Character

Q: Write a Python program that prints the following using a single print statement:

Line 1

Line 2

Line 3

Solution:

```
print("Line 1\nLine 2\nLine 3")
```

Question 9: Printing a Backslash

Q: Write a Python program that prints the following:

This is a backslash: \

Solution:

```
print("This is a backslash: \\")
```

Question 10: Suppressing the Newline

Q: Write a Python program that prints "Hello," and "World!" on the same line using two print statements.

Solution:

```
print("Hello,", end=" ")
print("World!")
```

Question 11: Add two numbers

Q: Write a Python program that adds two numbers and prints the output.

Solution:

```
print(4 + 3)
```

Questions to Practice:

1. Write a Python program that prints your Name and ID Number as the output.

2. Write a Python program that defines two variables, Course_title and Course_Code, and prints them in the format: "Name: [name], Age: [age]".

3. Write a Python program that prints "Computational Thinking for" and "Problem Solving" on the same line using string concatenation.

4. Write a Python program that uses the format method to print "Guido Van Rossum created Python Language".

5. Write a Python program that uses an f-string to print "Guido Van Rossum created Python Language".

6. Write a Python program for printing Multiple lines as following:
   Line1: First year Sem 1
   Line2: CTPS
   Line3: SEC 16
   Line4: Python

7. Write a Python program that prints using Escape Character the following **Faculty instructed, "Workbook is mandatory to bring for all the Lab Sessions"**

8. Write a Python program that prints the following using a single print statement using **Newline Character**:
   Line1: First year Sem 1
   Line2: CTPS
   Line3: SEC 16
   Line4: Python

9. Write a Python program that prints "Introduction to," and "Python" on the same line using two print statements

10. Write a Python program that adds two numbers and prints the output.

11. Write a Python program that prints the variable value

12. Write a Python program that increments the variable value by 5

13. **Create variables**

```
num_years = 4
days_per_year = 365
hours_per_day = 24
mins_per_hour = 60
secs_per_min = 60
```

Write a python program to Calculate number of seconds in four years.

Operators

1. **Arithmetic Operators**

Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, etc.

Examples of Arithmetic Operators

a = 10

b = 3

addition = a + b # Addition: 10 + 3 = 13

subtraction = a - b # Subtraction: 10 - 3 = 7

multiplication = a * b # Multiplication: 10 * 3 = 30

division = a / b # Division: 10 / 3 = 3.3333...

modulus = a % b # Modulus: 10 % 3 = 1 (remainder of the division)

exponentiation = a ** b # Exponentiation: 10^3 = 1000

floor_division = a // b # Floor Division: 10 // 3 = 3 (quotient without remainder)

2. **Assignment Operators**

Assignment operators are used to assign values to variables.

Examples of Assignment Operators

```
x = 5      # Simple assignment
x += 3     # Equivalent to x = x + 3 (x is now 8)
x -= 2     # Equivalent to x = x - 2 (x is now 6)
x *= 4     # Equivalent to x = x * 4 (x is now 24)
x /= 3     # Equivalent to x = x / 3 (x is now 8.0)
x %= 5     # Equivalent to x = x % 5 (x is now 3.0)
x **= 2    # Equivalent to x = x ** 2 (x is now 9.0)
x //= 3    # Equivalent to x = x // 3 (x is now 3.0)
```

3. **Comparison Operators**

Comparison operators compare two values and return a Boolean result (True or False).

Examples of Comparison Operators

a = 10

b = 20

print(a == b) # Equal: False

print(a != b) # Not equal: True

print(a > b)   # Greater than: False

print(a < b)   # Less than: True

print(a >= b) # Greater than or equal to: False

print(a <= b) # Less than or equal to: True

4. **Logical Operators**

Logical operators are used to combine conditional statements.

```
# Examples of Logical Operators
x = True
y = False
print(x and y) # Logical AND: False
print(x or y)  # Logical OR: True
print(not x)   # Logical NOT: False
```

5. **Membership Operators**

Membership operators are used to test if a sequence contains a specified value.

```
# Examples of Membership Operators
numbers = [1, 2, 3, 4, 5]

print(3 in numbers)   # True, because 3 is in the list
print(10 not in numbers) # True, because 10 is not in the list
```

6. **Bitwise Operators**

Bitwise operators operate on binary numbers at the bit level.

Examples of Bitwise Operators

a = 10 # Binary: 1010

b = 4   # Binary: 0100

print(a & b) # Bitwise AND: 1010 & 0100 = 0000 (0 in decimal)

print(a | b) # Bitwise OR: 1010 | 0100 = 1110 (14 in decimal)

print(a ^ b) # Bitwise XOR: 1010 ^ 0100 = 1110 (14 in decimal)

print(~a)   # Bitwise NOT: ~1010 = -1011 (-11 in decimal)

print(a << 1) # Left shift: 1010 << 1 = 10100 (20 in decimal)

print(a >> 1) # Right shift: 1010 >> 1 = 0101 (5 in decimal)

7. **Identity Operators**

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

Examples of Identity Operators

a = 10

b = 10

print(a is b) # True, because both a and b are the same object

print(a is not b) # False, because a and b are the same object

x = [1, 2, 3]

y = [1, 2, 3]

print(x is y) # False, because x and y are different objects with the same content

print(x is not y) # True, because x and y are different objects

Operator Precedence

1. Basic Arithmetic Operations

result = 2 + 3 * 4

print(result)

Explanation:

- Multiplication () has higher precedence than addition (+).

- So, 3 4 is evaluated first, giving 12.

- Then, 2 + 12 is evaluated, resulting in 14.

Output:

14

2. Using Parentheses to Change Precedence

result = (2 + 3) * 4

print(result)

Explanation:

- Parentheses () have the highest precedence, so 2 + 3 is evaluated first, giving 5.

- Then, 5 4 is evaluated, resulting in 20.

Output:

20

3. Mixed Operations with Exponentiation

result = 2 + 3 **2*2

print(result)

Explanation:

- Exponentiation () has higher precedence than multiplication () and addition (+).

- So, 3 2 is evaluated first, giving 9.

- Then, 9 2 is evaluated, resulting in 18.

- Finally, 2 + 18 is evaluated, resulting in 20.

Output:

20

4. Division and Multiplication

result = 8 / 4 * 2

print(result)

Explanation:

- Division (/) and multiplication () have the same precedence and are evaluated from left to right.

- So, 8 / 4 is evaluated first, giving 2.0.

- Then, 2.0 2 is evaluated, resulting in 4.0.

Output:

4.0

5. Mix of Addition, Multiplication, and Modulo

result = 5 + 3 * 4 % 6

print(result)

Explanation:

- The multiplication () and modulo (%) operations have higher precedence than addition (+).

- First, 3 * 4 is evaluated, resulting in 12.

- Next, 12 % 6 is evaluated, resulting in 0 (since 12 divided by 6 leaves a remainder of 0).

- Finally, 5 + 0 is evaluated, resulting in 5.

Output:

5

6. Exponentiation with Multiplication and Division

result = 2 ** 3 * 4 / 2

print(result)

Explanation:

- Exponentiation (**) has the highest precedence, so 2 ** 3 is evaluated first, resulting in 8.

- Then, multiplication (*) and division (/) are evaluated from left to right.

- 8 * 4 is evaluated next, resulting in 32.

- Finally, 32 / 2 is evaluated, resulting in 16.0.

Output: 16.0

7. Subtraction and Division with Parentheses

result = (20 - 5) / 3 + 2 * 4

print(result)

Explanation:

- Parentheses ( ) have the highest precedence, so 20 - 5 is evaluated first, giving 15.

- Then, 15 / 3 is evaluated, resulting in 5.0.

- Next, 2 * 4 is evaluated, giving 8.

- Finally, 5.0 + 8 is evaluated, resulting in 13.0.

Output:

13.0

8. Nested Parentheses and Exponentiation

result = 2 * (3 + 4) ** 2

print(result)

Explanation:

- Parentheses ( ) have the highest precedence, so 3 + 4 is evaluated first, giving 7.

- Then, exponentiation (**) is evaluated next, so 7 ** 2 is calculated, resulting in 49.

- Finally, 2 * 49 is evaluated, resulting in 98.

Output:

98

9. Complex Expression with Multiple Operations

result = 5 + 2 ** 3 * 4 - 8 / 2

print(result)

Explanation:

- The exponentiation (**) is evaluated first, so 2 ** 3 gives 8.

- Then, the multiplication (*) is evaluated, so 8 * 4 results in 32.

- The division (/) is evaluated next, so 8 / 2 results in 4.0.

- Finally, the addition (+) and subtraction (-) are evaluated from left to right:

  - 5 + 32 gives 37.

  - 37 - 4.0 results in 33.0.

Output:

33.0

10. Comparison with Logical AND

result = 5 > 3 and 2 < 4

print(result)

Explanation:

- Comparison operators (>, <) have higher precedence than logical operators like and.

- So, 5 > 3 is evaluated first, resulting in True.

- Then, 2 < 4 is evaluated, resulting in True.

- Finally, True and True is evaluated, which gives True.

Output:

True

11. Not Operator with Comparison

result = not 3 > 1

print(result)

Explanation:

- The not operator has lower precedence than comparison operators.

- So, 3 > 1 is evaluated first, resulting in True.

- Then, not True is evaluated, resulting in False.

Output:

False

Round( ) Function

Example 1: Rounding a Floating-Point Number

Example 1: Simple rounding

number = 5.67

rounded_number = round(number)

print(rounded_number) Output: 6

Explanation:

Here, 5.67 is closer to 6 than to 5, so the round() function rounds it up to 6.

Example 2: Rounding to a Specific Number of Decimal Places

Example 2: Rounding to 1 decimal place

number = 3.14159

rounded_number = round(number, 2)

print(rounded_number) Output: 3.14

Explanation:

In this example, the round() function rounds the number 3.14159 to two decimal places, resulting in 3.14.

Example 3: Rounding Halfway Cases

Example 3: Rounding a halfway case

number = 2.675

rounded_number = round(number, 2)

print(rounded_number) Output: 2.67

Explanation:

In Python, when rounding halfway cases (e.g., .5), it follows the "round to even" strategy (also known as "bankers' rounding"). Here, 2.675 rounded to two decimal places results in 2.67 instead of 2.68.

Example 4: Rounding a Negative Number

Example 4: Rounding a negative number

number = -1.27

rounded_number = round(number)

print(rounded_number) Output: -1

Explanation:

Negative numbers are rounded similarly to positive numbers. In this case, -1.27 is closer to -1 than to -2, so it rounds to -1.

Example 5: Rounding Large Numbers

Example 5: Rounding a large number

number = 123456.789

rounded_number = round(number, -3)

print(rounded_number) Output: 123000

Explanation:

Here, round() is used with a negative number of decimal places (-3), rounding 123456.789 to the nearest thousand, which results in 123000.

Conditional Statements

1.#if statement

```
a=7
b=10
if a>b:
    print("CTPS")
print("Python")
```

2. #if-else statements

```
age=17
if age>=18:
    print("eligible to vote")
else:
    print("not eligible to vote")
print("elections are over")
```

3. #if-elif-else statements

```
a=int(input())
b=int(input())
c=int(input())
if a>b and a>c:
    print("a is largest")
elif b>a and b>c:
    print("b is largest")
elif c>a and c>b:
    print("c is largest")
else:
    print("Welcome to Python")
```

4.#Nested if-else statements

```
a=4
if a>0:
    if a>=5:
        print("CTPS")
    else:
        print("Python")
else:
    print("Coding")
```

5. If-else conditional statement with Format string

```
name = input("Enter your name: ")
age = int(input("Enter your age: "))
if age >= 18:
    print(f"{name}, you are eligible to vote.")
else:
    print(f"Sorry {name}, you are not eligible to vote yet.")
```

6.

```
ch=int(input("enter marks of chemistry"))
b=int(input("enter marks of biology"))
m=int(input("enter marks of maths"))
p=int(input("enter marks of physics"))
c=int(input("enter marks of computers"))
total=ch+b+m+p+c
per=(total/500)*100
if per>=90:
    print("grade a")
elif per>=80 and per<90:
    print("grade b")
elif per>=70 and per<80:
    print("grade c")
```

```
elif per>=60 and per<70:
    print("grade d")
elif per>=40 and per<60:
    print("grade e")
else:
    print("fail")
```

7. Short hand if...else statement

```
x=5
print("x is greater") if x>5  else print("x is smaller or equal")
```

while loop statements

1.

```
#generate n natural numbers
i=1
n=int(input("enter the limit: "))
while(i<=n):
    print(i)
    i+=1
```

2.

```
#sum of n natural numbers
sum=0
count=1
while(count<10):
    sum=sum+count
    count+=1
    print(count)
    print(sum)
```

3.

```
#Fibonacci series
# Number of terms to be generated
num_terms = int(input("Enter the number of terms: "))
# Initialize the first two terms and a counter
a, b = 0, 1
count = 0
# Print the Fibonacci series
print("Fibonacci series:")
while count < num_terms:
    print(a)
    # Update the values of a and b for the next term
    a, b = b, a + b
    count += 1
```

4.

```
i = 1
while i < 6:
  print(i)
  if (i == 3):
    break
  i += 1
```

5.

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

for Loop statements

1.

```
fruit="apple"
for i in fruit:
    print(i)
```

2.

```
numbers=[1,2,3,4,5]
for i in numbers:
    print(i)
```

3.

```
name=input("enter a string: ")
for i in range(0,len(name)):
    print(name[i])
```

5.

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

6.

```
for i in range(0,5):
    print(i)
else:
    print("finally, else part corresponding statement.")
```

7.

```
for n in range(1,5):
    print(n)
```

8.

```
x=range(3,6)
for n in x:
    print(n)
```

9.

```
x=range(0,6,2)
for n in x:
    print(n)
```

10.

```
word="Python"
for char in word:
    print(char)
```

11.

```
numbers = [1, 2, 3, 4, 5]
#For loop with a break condition
for i in numbers:
    if i == 5:
        print("Found 5, stopping the loop.")
        break
    print(i)
```

12.

```
numbers = [1, 2, 3, 4, 5]
#For loop with a continue condition
for i in numbers:
    if i == 3:
        print("Skipping 3.")
        continue
    print(i)
```

13.

```
for i in range(0,5):
    if i==2:
        continue
    print(i)
```

14.

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
  for y in fruits:
    print(x, y)
```

Strings in Python

1. String Data Type

In Python, a string is a sequence of characters enclosed in quotes. Strings can be enclosed in single quotes (`'...'`), double quotes (`"..."`), or triple quotes (`'''...'''` or `"""..."""`).

Strings are immutable, meaning their content cannot be changed after they are created.

Example 1:

```
my_string = "Hello, Python!"
print(my_string)  # Output: Hello, Python!
```

Example 2:

```
multiline_string = """This is a multiline string.
It can span multiple lines.
Great for documentation or long texts."""
print(multiline_string)
```

Output:

```
This is a multiline string.
It can span multiple lines.
Great for documentation or long texts.
```

2. Indexing in Strings

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

What is an Array?

An array is a collection of items of same data type stored at contiguous memory locations.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets [ ] can be used to access elements of the string.

Indexing allows you to access individual characters in a string.

Python uses zero-based indexing, so the first character has an index of 0.

You can also use negative indexing to start from the end of the string.

Example1:

```
my_string = "Python"
print(my_string[0])   # Output: P
print(my_string[-1])  # Output: n
```

Example2:

```
word = "Programming"
first_letter = word[0]         # P
last_letter = word[-1]         # g
middle_letter = word[len(word)//2]  # a
print(first_letter, last_letter, middle_letter)
```

Output:

```
P g a
```

3. Iterating or Looping Over Strings

You can iterate over a string using a `for` loop to access each character individually.

Example1:

```
my_string = "Python"
for char in my_string:
    print(char)
```

Output:

```
P
y
t
h
o
n
```

Example2:

```
vowels = "aeiou"
sentence = "Python is awesome!"
vowel_count = 0
for char in sentence:
    if char.lower() in vowels:
        vowel_count += 1
print(f"Number of vowels: {vowel_count}")  #
```

Output: Number of vowels: 6

4. Strings Are Objects

In Python, strings are objects of the `str` class, which means they have methods and attributes that you can use.

Example1:

```
my_string = "Hello, World!"

print(type(my_string))
```

Output: <class 'str'>

Example2:

```
my_string = "hello"

capitalized = my_string.capitalize()

print(capitalized)
```

Output: Hello

5. String Methods

Python provides a wide variety of built-in string methods that allow you to manipulate and analyze strings.

Example1:

```
my_string = "Python programming"

print(my_string.upper())     # Output: PYTHON PROGRAMMING

print(my_string.lower())     # Output: python programming

print(my_string.replace("Python", "Java"))  # Output: Java programming
```

Example2:

```
text = "Python programming is fun! "

# Strip leading/trailing spaces

cleaned_text = text.strip()

# Split into words

words = cleaned_text.split()

# Join words with a hyphen

hyphenated = '-'.join(words)

print(hyphenated)
```

Output: Python-programming-is-fun!

6. String Slicing

Slicing allows you to extract a portion of a string. The general syntax for slicing is `string[start:stop:step]`.

- `start`: The starting index (inclusive).

- `stop`: The stopping index (exclusive).

- `step`: The step or interval (optional).

Example1:

```
my_string = "Hello, World!"

print(my_string[0:5])  # Output: Hello
```

Example2:

```
# Reverse a string using slicing
reversed_string = my_string[::-1]
print(reversed_string)  # Output: !dlroW ,olleH
# Extract every second character from the string
every_second_char = my_string[::2]
print(every_second_char)  # Output: Hlo ol!
```

7. Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

```
txt = "The best things in life are free!"
print("free" in txt)   #Output: True
```

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

```
txt = "The best things in life are free!"
print("expensive" not in txt)   #Output: True
```