UNIT-1

1. Which character is used in Python to make a single-line comment?

In Python, the hash (`#`) character is used to indicate a single-line comment.

2. What will be the output of `print(type(2**5))` in Python?

The expression `2**5` calculates 2 raised to the power of 5, which is 32. The `type()` function returns the type of the argument. Therefore, the output will be:

```
<class 'int'>
```

3. What will be the output of `print(type("LJU"))` in Python?

The expression `"LJU"` is a string, so the output will be:

```
<class 'str'>
```

4. What will be the output of `print(type(3*5/5))` in Python?

The expression `3*5/5` evaluates to 3.0, which is a float. Therefore, the output will be:

```
<class 'float'>
```

5. If `x=3.123`, then `int(x)` will give?

Converting a float to an integer using `int()` truncates the decimal part. So, `int(x)` will give:

```
3
```

6. Which one of the following is the correct way of declaring and initializing a variable, `x` with the value 5?

```
x = 5
```

11. What happens when `'2' == 2` is executed?

The expression compares a string (`'2'`) with an integer (`2`). They are of different types, so the result is `False`.

12. What will be the output of this program?

```
_ = '1 2 3 4 5 6'
print(_)
```

The output will be:

```
1 2 3 4 5 6
```

13. The following is displayed by a print function call. Select all of the function calls that result in this output.

```
tom
sam
harry
```

All three names can be printed using the following function calls:

```
print('tom\nsam\nharry')
```

14. What will be the output of the following program on execution?

```
print(print(print("python")))
```

The output will be:

```
python
None
None
```

Explanation: The inner `print("python")` prints "python" and returns `None`. This `None` is then printed by the outer `print()` functions.

15. Which is the correct operator for power (X^y)?

The correct operator for exponentiation is `**`. So, `X**y` represents X raised to the power of y.

16. What is the answer to this expression, `34 % 3`?

The `%` operator represents the modulus, which gives the remainder of the division. The remainder of `34 / 3` is 1. Therefore, the answer is `1`.

17. What will be the output of the following program on execution?

```
a = 0
b = 6

x = (a or b) or ((a and a) or (a and b))
print(x)
```

The output will be `6`. This is because `(a or b)` evaluates to `6` (the first non-zero value encountered).

18. What will be the output of the following program on execution?

```
a = 0
b = 6

x = (a or b) or ((a and a) or (a and b))
y = not(x)
print(y)
```

The output will be `False`. The `not` operator negates the truth value, and since `x` is non-zero (True), `not(x)` is `False`.

19. What will be the output of this program?

```
print(True ** False / True)
```

The output will be `1.0`. `True` is equivalent to `1` and `False` is equivalent to `0`.

20. What is the average value of the following Python code snippet?

```
grade1 = 80
grade2 = 90
average = (grade1 + grade2) / 2
```

The average value is `(80 + 90) / 2`, which is `85.0`.

21. Which of the following errors occurs when you execute the following Python code?

```
apple = mango
```

The error is a `NameError` because `mango` is not defined.

22. What is the output of this expression, `3*1**3`?

The expression is equivalent to `3 * (1**3)`, and any number raised to the power of 1 is itself. Therefore, the output is `3`.

25. Which of the following is not a comparison operator in Python?

`=`, which is an assignment operator, is not a comparison operator.

26. What will be the output of the following program on execution?

```
a = 4
b = 6
c = 3
print(a + b * c / a - b)
```

The output will be `2.5`. The expression follows the order of operator precedence.

27. What will be the value of the following Python expression?

```
8 + 2 % 3
```

The output will be `10`. The modulus (`%`) has higher precedence than addition, so `2 % 3` is evaluated first, resulting in `2`, and then added to `8`.

28. What will be the value of `x` in the following Python expression?

```
x = int(63.55 + 8 / 3)
```

The output will be `66`. The expression inside `int()` is evaluated first, and then the result

is converted to an integer.

29. What are the values of the following Python expressions?

```
2**(3**2)
(2**3)**2
2**3**2
```

The values are:

```
512
64
512
```

30. What will be the output of this program?

```
print(6 + 5 - 4 * 3 / 2 % 1)
```

The output will be `9.0`. The expression follows the order of operator precedence.

31. "What will be the output of this program?

```
print(int(6 == 6.0) * 3 + 4 % 5)"
```

Answer: `7` **Explanation:**

- `6 == 6.0` evaluates to `True`, and when cast to an integer, it becomes `1`.
- `1 * 3` is `3`.
- `4 % 5` is `4`.
- Adding them together: `3 + 4` equals `7`.

32. "What will be the value of X in the following Python expression?

```
X = 2+9*((3*12)-8)/10"
```

Answer: `27.2` **Explanation:**

- Inside the parentheses: `3 * 12` is `36`, and then subtracting `8` results in `28`.

- The expression becomes `2 + 9 * 28 / 10`, which is `27.2`.

33. "What will be the output of the following Python code?

```
new = (1 and ""True"") and ('False' or Train)
str = 'This statement is ' + new
print(""This is False"" if ""False"" in new else ""This is True"")"
```

Answer: `"This is False"` **Explanation:**

- `(1 and ""True"")` evaluates to `""True""`.
- `('False' or Train)` evaluates to `""False""` (the string, not the boolean `False`).
- `new` becomes `""True"" and ""False""`, and the string comparison checks if `""False""` is in `new`.
- Since it is, the output is `"This is False"`.

34. "What will be the value of the following Python expression?

```
8 + 1 % 3"
```

Answer: `9` **Explanation:**

- The expression is evaluated as `8 + (1 % 3)`, and the modulus operation `%` results in `1`, so the final value is `9`.

35. "What should be the output of the following Python code snippet:

```
x=0.0
y=48>0
z=11<7
print(not(float(x or y or z)))"
```

Answer: `False` **Explanation:**

- `x or y or z` evaluates to `True` because `0.0 or True or False` is considered `True`.
- `not(float(Ture))` to `False`. The `print()` statement prints `False`.

36. "What will be the output of the following Python function in jupyter notebook?

```
""a""+""bc""""
```

Answer: `"abc"` **Explanation:**

- The `+` operator concatenates the two strings, resulting in `"abc"`.

37. "What is the order of precedence in python?

i) Parentheses

ii) Division

iii) Multiplication

iv) Exponential

v) Addition

vi) Subtraction" **Answer:** i) Parentheses, iv) Exponential, iii) Multiplication, ii) Division, v) Addition, vi) Subtraction **Explanation:** In Python, operations inside parentheses have the highest precedence, followed by exponentiation, multiplication, division, addition, and subtraction.

38. "The following is displayed by a print function call. Select all of the function calls that result in this output.

```
tom

sam

harry"
```

Answer:

```
print('''tom
\nsam
\nharry''')
```

39. "What will be the output of the following program on execution?

```
x = Str(""Python is a very\b simple\bsubject"")
print(x)"
```

Answer: `NameError` **Explanation:** The correct function name is `str`, not `Str`.

40. "What will be the output of the following program.

```
x = int(43.55+4/6)
print(X)"
```

Answer: `NameError` **Explanation:** The variable is defined as `x`, but it is printed as `X`. The corrected version would be:

```
x = int(43.55 + 4 / 6)
print(x)
```

41. "What is the output of the following assignment operator

```
a = 10
b = a -= 2
print(b)"
```

Answer: `SyntexError` **Explanation:** The expression `a -= 2` is an in-place assignment and does not return a value. Trying to assign it to `b` will result in a `SyntaxError`.

42. "What will be the output of the following program on execution?

```
a = 4
b = 6
c = 3
d = 2
print(a + d ** b * c / a - b)"
```

Answer: `46.0` **Explanation:** The expression follows the order of operator precedence.

43. "What will be the output of this program?

```
print(int(""6"" == 6.0) * 3 + 4 % 5)"
```

Answer: `4` **Explanation:**

- `""6"" == 6.0` evaluates to `True`.
- When cast to an integer, it becomes `1`.
- `1 * 3` is `3`.
- `4 % 5` is `4`.
- Adding them together: `3 + 4` equals `4`.

44. "What will be the output of the following program on execution?

```
a = 0
b = 6
c = 9
d = 10
x = (a or b) and ((a or c) or (b and d))
print(x)"
```

Answer: `9` **Explanation:** The expression evaluates to `6 and (9 or 10)` and gives the answer after `6 and 9` as `9`.

45. "What will be the value of X in the following Python expression?

```
X = 2 + 9 * ((3 * 12) - 8) / 10
print(bool(X))"
```

Answer: `True` **Explanation:** The expression evaluates to a non-zero value, making the boolean result `True`.

46. "What will be the datatype of the var in the below code snippet?

```
var = 10
print(type(var))
```

```
var = ""Hello""
print(type(var))"
```

Answer: `int` and `str` **Explanation:** The variable `var

`is initially assigned an integer (10) and later reassigned a string ("Hello"`).

47. "What will be the output of this program?

```
print(True * False / True)"
```

Answer: `0.0` **Explanation:** `True` is equivalent to `1` and `False` is equivalent to `0`. The result is `0.0`.

48. "What is the output of the following python code:

```
x = 125
y = 13
x //= y
print(x)"
```

Answer: 9 **Explanation:** `x //= y` performs floor division and assigns the result to `x`, so `125 // 13` is 9.

49. "What is the output of the following Python code:

```
print(bool(0), bool(3.14159), bool(-3), bool(False))"
```

Answer: `False True True False` **Explanation:**

- `bool(0)` is `False`.
- `bool(3.14159)` is `True`.
- `bool(-3)` is `True`.
- `bool(False)` is `False`.

50. "a=5 b=10 c=1 print(a**c, b//a, c%a)"

Answer: `5 2 1` **Explanation:**

- `a**c` is `5**1`, which is 5.
- `b//a` is `10//5`, which is 2.
- `c%a` is `1%5`, which is `1`.

52. What is the output of this expression, `3**1**3/True`?

Answer: `3.0` **Explanation:** The expression is equivalent to `3 ** (1 ** 3) / True`, and any number raised to the power of 1 is itself. Therefore, the output is `3.0`.

53. "What will be the output of the following program on execution?

```
a = 0
b = 5
a or b == 5 or True + 7 - 4 * 3"
```

Answer: `True` **Explanation:** The expression is evaluated as `0 or (5 == 5) or (True + 7 - 4 * 3)`, and `True + 7 - 4 * 3` is `True`.

54. "What will be the output of the following program on execution?

```
a = 50
b = 60
print((a and b) / False)"
```

Answer: `ZeroDivisionError` **Explanation:** Division by `False` raises a `ZeroDivisionError`.

UNIT-2

67. "What does the following code print?

```
if 4 + 5 == 10:
    print("TRUE")
else:
    print("FALSE")
print("TRUE")"
```

Answer:

```
FALSE
TRUE
```

Explanation: The `if` condition is `False`, so it executes the code block under `else`, printing "FALSE." The `if` block is skipped, and then "TRUE" is printed outside the if-else structure.

68. "What does the following code print?

```
x = -10
if x < 0:
    print("The negative number ", x, " is not valid here.")
print("This is always printed")"
```

Answer:

```
The negative number -10  is not valid here.
This is always printed
```

Explanation: The `if` condition is `True`, so it executes the code block under `if`, printing the message. The following `print` statement is always executed.

69. "Which of the following is true about the code below?

```
x = 3
if (x > 2):
    x = x * 2;
if (x > 4):
    x = 0;
print(x)"
```

Explanation: The first `if` condition is `True`, so `x` is multiplied by 2. Then, the second `if` condition is `True` (as `x` is now 6), so `x` is set to 0. The final value of `x` is 0, and it is printed.

71. What keyword would you use to add an alternative condition to an if statement?

Answer: `elif`

72. "Which of the following will evaluate to true?

I. True and False

II. False or True

III. False and (True or False)"

Answer:

```
I. False
II. True
III. False
```

Explanation:

- I. `True and False` is `False`.
- II. `False or True` is `True`.
- III. `False and (True or False)` is `False`.

74. "What is the output from the following code?

```
a = 3
b = (a != 3)
print(b)"
```

Answer: `False` **Explanation:** `(a != 3)` evaluates to `False`, and the result is assigned to `b`. The `print` statement prints `False`.

75. Which of the following evaluates to True when a is equal to b or when a is equal to 5?

Answer: `a == b or a == 5`

76. Which statement will check if a is equal to b?

Answer: `if a == b:`

77. "Given the nested if-else structure below, what will be the value of x after code execution completes

```
x = 0
a = 0
b = -5
if a > 0:
    if b < 0:
        x = x + 5
    elif a > 5:
        x = x + 4
    else:
        x = x + 3
else:
    x = x + 4
print(x)"
```

Answer: `4` **Explanation:** The outer `if` condition (`a > 0`) is `False`, so the `else` block is executed, setting `x` to `x + 4`.

78. "Given the nested if-else below, what will be the value x when the code executed successfully

```
x = 2
a = 5
b = 5
if a > 0:
    if b < 0:
        x = x + 5
    elif a > 5:
        x = x + 4
    else:
        x = x + 3
else:
    x = x + 2
print(x)"
```

Answer: `5` **Explanation:** The outer `if` condition (`a > 0`) is `True`, and the inner `elif` condition (`a > 5`) is also `True`. So, `x` is incremented by `4`.

79. "What is the output of the following if statement

```
a, b = 12, 5
if a + b:
    print('True')
else:
    print('False')"
```

Answer: `True` **Explanation:** The expression `a + b` evaluates to `17`, which is considered `True` in a boolean context.

81. A loop becomes an infinite loop if a condition never becomes _____.

Answer: `False`

82. If the else statement is used with a while loop, the else statement is executed when the condition becomes _____.

Answer: `False`

83. Python programming language allows the use of one loop inside another loop known as?

Answer: Nested loop

84. "What will be the output of the given Python code?

```
n = 7
c = 0
while n:
    if n > 5:
        c = c + n - 1
        n = n - 1
    else:
        break
print(n)
print(c)"
```

Answer:

```
5
11
```

Explanation: The `while` loop continues as long as `n` is non-zero. Inside the loop, when `n > 5`, `c` is updated, and `n` is decremented. When `n <= 5`, the loop breaks. The final values are printed.

85. "How many times will the loop run?

```
i = 2
while i > 0:
    i = i - 1"
```

Answer: The loop will run 2 times.

86. "How many times will the condition be checked?

```
i = 2
while i > 0:
    i = i - 1"
```

Answer: The condition will be checked 3 times.

87. "What will be the output of the following Python code?

```
i = 1
while True:
    if i % 3 == 0:
        break
    print(i)
        i += 1"
```

Answer:

```
Error
```

Explanation: It will give Indentation Error in line 6 --> i + = 1

88. "What is the value of the var after the for loop completes its execution

Let's analyze the code step by step:

```
var = 10
for i in range(10):
    for j in range(2, 10, 1):
        if var % 2 == 0:
            continue
            var += 1
    var += 1
else:
    var += 1
print(var)
```

1. `var` is initialized to `10`.
2. The outer loop (`for i in range(10)`) runs 10 times.

3. The inner loop (`for j in range(2, 10, 1)`) runs for each iteration of the outer loop. However, the `continue` statement is encountered if `var` is even, which skips the increment (`var += 1`) inside the inner loop.
4. If the `continue` statement is not triggered, `var` is incremented by 1.
5. After the inner loop, there is an `else` block associated with the outer loop. This block is executed once after the completion of the loop unless the loop was terminated by a `break`.
6. In the `else` block, `var` is incremented by 1.
7. Finally, `print(var)` prints the value of `var`.

Let's calculate the final value of `var`:

- In each iteration of the outer loop, `var` is incremented by 1.
- If `var` is even, the increment inside the inner loop is skipped.
- The `else` block is executed once after the outer loop.

Since the `continue` statement inside the inner loop prevents the increment when `var` is even, the final value of `var` is increased by the total number of iterations of the outer loop (10 times) and the `else` block (once).

Therefore, the final value of `var` is `10 + 10 + 1 = 21`.

So, the output of the code is `21`.

89. "What will be the output of the following Python code?

```
for i in range(0, 2, -1):
    print("Hello")"
```

Answer: No output (The loop won't run) **Explanation:** The loop will not run because the start value (`0`) is greater than the end value (`2`) when the step is negative.

91. "Which of the following sequences would be generated by the given line of code?

```
range(5, 0, -2)"
```

Answer: `5 3 1` **Explanation:** The `range(5, 0, -2)` generates values starting from `5` (inclusive), decreasing by `2` until reaching `0` (exclusive).

92. When does the else statement written after a loop execute?

Answer: The `else` statement after a loop executes when the loop condition becomes `False` or when the loop is exhausted.

93. "What is the output of the following for loop and range() function

```
for num in range(-2, -5, -1):
    print(num, end=",")
```

Answer: `-2,-3,-4` **Explanation:** The `range(-2, -5, -1)` generates values starting from `-2` (inclusive), decreasing by `1` until reaching `-5` (exclusive).

94. "What will be the output of the following code?

```
x = 12
for i in x:
    print(i)"
```

Answer: `TypeError` **Explanation:** `for i in x` will raise a `TypeError` because `x` is an integer, and integers are not iterable.

95. "What is the value of x after the following nested for loop completes its execution

```
x = 0
for i in range(10):
    for j in range(-1, -10, -1):
        x += 1
        print(x)
```

1. `x` is initialized to `0`.
2. The outer loop (`for i in range(10)`) runs 10 times.
3. The inner loop (`for j in range(-1, -10, -1)`) runs for each iteration of the outer loop. It runs 9 times, decrementing `j` from `-1` to `-9`.
4. Inside the inner loop, `x` is incremented by 1 for each iteration.
5. The `print(x)` statement prints the current value of `x`.

Now, let's see how many times `x` is incremented:

For each iteration of the outer loop (10 times), the inner loop runs 9 times. So, `x` is incremented by a total of `10 * 9 = 90` times.

The output will be the values of `x` printed during these increments.

Therefore, the output will be a sequence of numbers starting from `1` and ending at `90`.

Here's a part of the output:

```
1
2
3
...
90
```

So, the final value of `x` is `90`.

96. "What is the output of the following `range()` function?

```
for num in range(2, -5, -1):
    print(num, end=",")
```

Answer: `2, 1, 0, -1, -2, -3, -4,` **Explanation:** The `range(2, -5, -1)` generates values starting from `2` (inclusive), decreasing by `1` until reaching `-5` (exclusive).

97. "What is the value of `x`?

```
x = 0
while (x < 100):
    x += 2
print(x)"
```

Answer: `100` **Explanation:** The `while` loop increments `x` by `2` in each iteration until `x` becomes greater than or equal to `100`.

98. "What is the output of the following nested loop?

```
for num in range(10, 14):
    for i in range(2, num):
        if num % i == 1:
            print(num)
            break
```

Answer: `10 11 12 13` **Explanation:** The outer loop iterates over the numbers from 10 to 13 (inclusive). For each number, the inner loop iterates over the numbers from 2 to the current number (exclusive). Inside the inner loop, there is an if statement that checks if the current number is divisible by the current iterator. If it is, the current number is printed and the `break` statement is executed, which causes the inner loop to terminate immediately.

Here is a more detailed explanation of each part of the code:

- `for num in range(10, 14)`: This loop iterates over the numbers from 10 to 13 (inclusive). The variable `num` will take on each of these values in turn.
- `for i in range(2, num)`: This loop iterates over the numbers from 2 to the current value of `num` (exclusive). The variable `i` will take on each of these values in turn.
- `if num % i == 1`: This if statement checks if the current value of `num` is divisible by the current value of `i`. If it is, the `if` statement block is executed.
- `print(num)`: This statement prints the current value of `num`.
- `break`: This statement causes the current loop (the inner loop) to terminate immediately.

The overall effect of the code is to print the prime numbers from 10 to 13. A prime number is a number that has exactly two factors: 1 and itself. The `break` statement is used to prevent the

inner loop from continuing to iterate after a factor has been found, which saves time and makes the code more efficient.

Here is the output of the code:

```
10
11
12
13
```

99. "What will be the output of the following Python code?

```
i = 1
while True:
    if i % 3 == 0:
        break
    print(i)
    i += 1
```

Answer: `1  2` **Explanation:** The `while True` creates an infinite loop, but the `break` statement is encountered when `i % 3 == 0`, so the loop exits.

100. "What will be the output of the following Python code?

Let's analyze the code:

```
i = 1
while True:
    if i % 2 == 0:
        break
    print(i)
    i += 2
```

1. The variable `i` is initialized to `1`.
2. The `while True` creates an infinite loop.
3. Inside the loop, it checks if `i % 2 == 0`. If `i` is even, it breaks out of the loop.
4. It prints the current value of `i` and increments `i` by `2`.

Since `i` is initially `1` (an odd number), the condition `i % 2 == 0` is not satisfied in the first iteration. Therefore, the loop continues to run indefinitely, printing odd numbers.

The output of the code will be an infinite sequence of odd numbers:

```
1
3
5
7
9
... (continues indefinitely)
```

The loop will not terminate because the break condition is not met for odd values of `i`.

101. "What will be the output of the following Python code?

```
i = 2
while True:
    if i % 3 == 0:
        break
    print(i)
    i += 2
```

Answer:

```
2
4
```

Explanation: The code initializes the variable i to 2. The while loop iterates as long as the condition True is true, which is always the case. Inside the loop, the value of i is printed to the console. The value of i is then incremented by 2. The if statement checks if the value of i is divisible by 3. If it is, the break statement is executed, which causes the loop to terminate immediately. In this case, the condition i % 3 == 0 is not true for the first two iterations of the loop, so the values 2 and 4 are printed to the console. However, on the third iteration, the condition i % 3 == 0 is true, so the loop terminates and no further output is generated.

102. "What will be the output of the following Python code?

```
i = 1
while False:
    if i % 2 == 0:
        break
    print(i)
    i += 2
```

Answer: No output (The loop won't run) **Explanation:** The `while False` condition makes the loop never execute.

103. "What will be the output of the following Python code?

```
True = False
while True:
    print(True)
    break
```

Answer: `SyntaxError: cannot assign to True` **Explanation:** Attempting to assign a value to the built-in constant `True` raises a `SyntaxError`.

104. "What will be the output of the following Python code?

```
for i in range(0):
    print(i)
```

Answer: No output (The loop won't run) **Explanation:** The `range(0)` generates an empty sequence, so the loop won't run.

105. "What will be the output of the following Python code?

```
for i in range(2.0):
    print(i)
```

Answer: `TypeError` **Explanation:** `range()` requires integer arguments, so attempting to use a `float` raises a `TypeError`.

106. "What will be the output of the following Python code?

```
for i in range(int(2.0)):
    print(i)
```

Answer: `0 1` **Explanation:** `int(2.0)` results in `2`, so the loop iterates from `0` to `1`.

107. "What will be the output of the following Python code?

```
for i in range(float('inf')):
    print(i)
```

Answer: TypeError: 'float' object cannot be interpreted as an integer in range()

108. "What will be the output of the following Python code?

```
for i in range(5):
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

Answer: `0 1 2 3 4 Here` **Explanation:** The loop runs for `i` in the range `0` to `4`, and the `else` block is executed because the loop completes without encountering a `break` statement.

109. "What will be the output of the following Python code?

```
for i in range(10):
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

Answer: `0 1 2 3 4` **Explanation:** The loop runs for `i` in the range `0` to `9`, and the `else` block is not executed because the loop is terminated by the `break` statement when `i == 5`.

110. "What will be the output of the following Python code snippet?

```
x = 2
for i in range(x):
    x -= 2
    print(x)
```

1. The variable `x` is initialized to `2`.
2. The `for i in range(x):` loop iterates twice because `range(x)` creates a sequence `[0, 1]`.
3. Inside the loop, `x` is decremented by `2` in each iteration.
4. It prints the current value of `x`.

Here's the breakdown:

- In the first iteration, `x` becomes `0`, and it prints `0`.
- In the second iteration, `x` becomes `-2`, and it prints `-2`.

The output of the code will be:

```
0
-2
```

111. Output of the Python code snippet:

```
x = 2
for i in range(x):
    x += 1
    print (x)
```

Explanation: The loop runs twice (for i = 0 and i = 1). In each iteration, `x` is incremented by 1, and the updated value of `x` is printed.

```
3
4
```

112. Output of the Python code:

```
i = 0
while i < 5:
    print(i)
    i += 1
    if i == 3:
        break
else:
    print(0)
```

Explanation: The `while` loop runs until `i` is less than 5. It prints the value of `i` in each iteration. If `i` becomes 3, the loop is terminated with a `break` statement, and the `else` block is not executed.

```
0
1
2
```

113. Output of the Python code:

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print(0)
```

Explanation: Similar to the previous example, the `while` loop runs until `i` is less than 3. It prints the value of `i` in each iteration. Since the condition becomes False after `i` reaches 3, the `else` block is executed once.

```
0
1
2
0
```

114. Output of the Python code:

```
x = 2
for i in range(x):
    x -= 2
    print (x)
```

Explanation: The loop runs twice (for i = 0 and i = 1). In each iteration, `x` is decremented by 2, and the updated value of `x` is printed.

```
0
-2
```

115. The `continue` statement can be used in:

- Loops (for and while)

116. Output of the Python code:

```
int("Enter value of x:")
for in range[0, 10]:
    print("They are equal")
```

```
else:
    print("They are unequal")
```

Explanation: There are syntax errors in this code. `int("Enter value of x:")` is not assigned to a variable, and there's a typo in the `for` loop syntax. It will result in a `SyntaxError`.

117. Output of the Python code:

```
a = 5
b = 5.0
print('yes') if (a == b) else 'no'
```

Explanation: This code uses a ternary conditional expression. It prints 'yes' if `a` is equal to `b`, otherwise 'no'.

```
yes
```

118. Output of the Python code:

```
a = b = 0
if (a = b):
    print(0)
else:
    print('otherwise')
```

Explanation: This code has a syntax error. The assignment inside the `if` condition should use `==` for comparison, not `=`. It will result in a `SyntaxError`.

119. Output of the Python code:

```
Step = 3
for e in range(0, step):
    if e%2==0:
        print('hello')
    else:
        print('goodbye')
```

Explanation: There's a typo in this code. The variable name `step` is not defined; it should be `Step`.

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

120. Output of the Python code:

```
x = 123
for i in x:
    print(i)
```

Explanation: This code has a `TypeError` because you cannot iterate over an integer (`x`). The `for` loop expects an iterable, and integers are not iterable.

```
TypeError
```

121. Output of the Python code snippet:

```
theSum = 0
for count in range(2, 11, 2):
    theSum += count
print(theSum)
```

Explanation: This code calculates the sum of even numbers from 2 to 10 (inclusive).

```
30
```

122. Output of the Python code snippet:

```
a = True
b = False
c = False

if not a or b:
    print (1)
elif not a or not b and c:
    print (2)
elif not a or b or not b and a:
    print (3)
else:
    print (4)
```

Explanation: This code uses conditional statements (`if`, `elif`, `else`). It checks multiple conditions and prints the corresponding number.

```
3
```

123. Output of the Python code:

```
var = 10
for i in range(5):
    for j in range(2, 3, 1):
        if var%2 == 0:
            break
        var += 1
    var += 1
else:
    var += 1
print(var)
The output of the Python code is 20.
```

Explanation:

The code initializes the variable `var` to 10. The first `for` loop iterates 5 times. Inside the first `for` loop, the second `for` loop iterates 1 time. Inside the second `for` loop, the condition `var % 2 == 0` is checked. If the condition is true, the `break` statement is executed, which causes the loop to terminate immediately. In this case, the condition is never true, so the loop completes all 1 iteration. The value of `var` is then incremented by 1. The first `for` loop then iterates to the next iteration.

After the first `for` loop completes, the `else` block is executed. Inside the `else` block, the value of `var` is incremented by 1. Finally, the value of `var` is printed to the console.

Here is a table summarizing the execution of the code:

Iteration	Action	Value of var
1	Initialize var to 10	10
2	Enter first for loop	10
3	Enter second for loop	10
4	Check condition var % 2 == 0 (False)	10
5	Increment var by 1	11
6	Exit second for loop	11
7	Increment var by 1	12
8	Check condition var % 2 == 0 (False)	12
9	Increment var by 1	13
10	Exit second for loop	13
11	Increment var by 1	14
12	Check condition var % 2 == 0 (False)	14
13	Increment var by 1	15
14	Exit second for loop	15
15	Increment var by 1	16
16	Check condition var % 2 == 0 (False)	16
17	Increment var by 1	17
18	Exit second for loop	17
19	Increment var by 1	18
20	Check condition var % 2 == 0 (False)	18
21	Increment var by 1	19
22	Exit second for loop	19
23	Increment var by 1	20
24	Exit first for loop	20
25	Print value of var	20

I hope this helps! Let me know if you have any other questions.
14

124. Output of the Python code:

```
A = 70
if A > 90:
    print('Grade A')
elif A > 70 and A < 90:
    print('Grade B')
elif A > 50 and A < 70:
    print('Grade C')
elif A > 35 and A < 50:
    print('Grade D')
else:
    print('Fail')
```

Explanation: This code determines the grade based on the value of `A`. Since `A` is 70, it falls into the second `else` condition as A>70 and A<70 condition is only given, and 'Fail' is printed.

```
Fail
```

125. Output of the Python code:

```
seconds = 3650
if seconds >= 3600:
    hour = seconds // 3600
    seconds %= 3600
    print(hour, "hours", end="")
if seconds >= 60:
    minute = seconds // 60
    seconds %= 60
    print(minute, "minutes", end="")
if seconds > 0:
    print(seconds, "seconds")
```

Explanation: This code converts the `seconds` variable into hours, minutes, and seconds and prints them.

```
1 hours 50 seconds
```

126. Output of the Python code:

```
A = 0
for i in range(4):
    if i % 2 == 0:
        pass
    else:
        continue
        break
    A += 1
print(A)
```

Explanation: This code uses `pass` to do nothing for even values of `i`. The `continue` statement is encountered for odd values, which skips the `break` statement. Therefore, the loop completes 2 times and gets skipped 2 times, and `A` is incremented twice.

```
2
```

127. Output of the Python code snippet:

```
count = 0
while(True):
    if count % 3 == 0:
        print(count, end=" ")
    if count > 15:
        break;
    count += 1
```

Explanation: This code prints numbers divisible by 3 until `count` exceeds 15.

```
0 3 6 9 12 15
```

128. Output of the Python code:

```
a = True
b = False
c = True

if not a or b:
    print("a")
elif not a or not b and c:
    print("b")
elif not a or b or not b and a:
    print("c")
else:
    print("d")
```

Explanation: This code uses conditional statements to determine which block to execute. The conditions are evaluated, and the first condition that is true is executed, which is elif not a or not b and .

```
b
```

129. Output of the Python code:

```
if 5 + 5 == 10:
    print("TRUE")
else:
    print("FALSE")
print("TRUE")
```

Explanation: The first block of code checks if 5 + 5 is equal to 10 and prints "TRUE" if true; otherwise, it prints "FALSE". The second `print` statement is executed regardless of the outcome of the `if` condition.

```
TRUE
TRUE
```

130. Which of the following will evaluate to true?

```
I. True and False
II. False or True
III. False and (True or False)
IV. True and (True or False)
```

Explanation: I. False

II. True

III. False

IV. True

Therefore, II and IV will evaluate to true.

131. Output of the nested loop:

```
for num in range(26, 30):
    for i in range(2, num):
        if num % i == 1:
            print(num, end=',')
            break
```

Output: The correct output is `26, 27, 28, 29,`.

Explanation:

1. The outer loop (`for num in range(26, 30)`) iterates through the numbers from 26 to 29.
2. The inner loop (`for i in range(2, num)`) checks if the current `num` is divisible by any number in the range from 2 to `num - 1`.
3. If `num` is not divisible by any number in that range, it is a prime number, and the loop will print it, appending a comma after each prime number.
4. The `break` statement ensures that only the first divisor found is considered for each `num`.

So, the correct output is indeed `26, 27, 28, 29,`.

132. Value of `var` after the for loop:

```
var = 10
for i in range(10):
```

```
    for j in range(2, 10, 1):
        if var % 2 == 0:
            var += 1
            continue
    var += 1
else:
    var += 1
print(var)
```

The value of var after the loop completes its execution is 31.

Here is a table summarizing the execution of the code:

Line	Code	Action	Value of var
1	var = 10	Initialize var to 10	10
2	for i in range(10):	Start first for loop	10
3	for j in range(2, 10, 1):	Start second for loop	10
4	if var % 2 == 0:	Check if var is even	10
5	var += 1	Increment var by 1	11
6	continue	Skip to the next iteration of the second for loop	11
7	for j in range(2, 10, 1):	Start second for loop again	11
8	if var % 2 == 0:	Check if var is even	11
9	var += 1	Increment var by 1	12
10	continue	Skip to the next iteration of the second for loop	12
11	...	...	...
20	var+=1	Increment var by 1	21
21	else:	Execute else block because the first for loop condition is false	21
22	var+=1	Increment var by 1	22
23	print(var)	Print the value of var	31

133. Output of the Python code snippet:

```
a = 5
b = 7
c = 2

if a > b:
    a, b = b, a
```

```
if a > c:
    a, c = c, a
if b > c:
    b, c = c, b

print(a, b, c, end="")
```

Output: The output will be the sorted values of `a`, `b`, and `c` separated by no space or any delimiter and comma at the end. 2 5 7,

134. Value of `x` after the nested for loop:

Let's analyze the given code:

```
x = 0
for i in range(1, 10):
    for j in range(-1, -10, -1):
        x += 1
        print(x)
```

The outer loop (`for i in range(1, 10):`) iterates from 1 to 9, and for each iteration, the inner loop (`for j in range(-1, -10, -1):`) iterates from -1 to -9 in reverse order.

In each iteration of the inner loop, `x` is incremented by 1, and the value of `x` is printed. However, since the inner loop iterates from -1 to -9, it will execute 9 times for each iteration of the outer loop.

So, the total number of times `x` is incremented and printed is 9 times (inner loop) * 9 times (outer loop) = 81 times.

The output will be the numbers from 1 to 81 (inclusive) printed on separate lines. The final value of `x` after the loop completes will be 81.

```
1
2
3
4
5
6
7
8
9
... (output continues up to 81)
```

Therefore, the value of `x` after the nested for loop completes its execution is 81.

135. Value of `x`:

```
x = 0
while x < 100:
    x += 3
print(x)
```

Explanation: The provided code snippet initializes the variable `x` to 0 and then enters a `while` loop that iterates as long as the condition `x < 100` is true. Inside the loop, the value of `x` is incremented by 3. After the loop terminates, the value of `x` is printed to the console.

Since the loop iterates until `x` reaches a value of 100, and `x` is incremented by 3 each iteration, the final value of `x` is 102. Therefore, the output of the code is:

```
102
```

136. Output of the program:

```
if (9 < 0) and (0 < -9):
    print("hello")
elif (9 > 0) or False:
    print("good")
else:
print("bad")
```

Output: IndentationError in last line of code

137. Output of the code:

```
c = 1
s = 0
while c <= 8:
    c = c - 1
    s = s + c
    c = c + 2
print(s)
```

Output:

1. Initialize variables:
   - `c` is initialized to 1
   - `s` is initialized to 0
2. Enter the `while` loop:
   - The condition `c <= 8` is evaluated. Since `c` is initially 1 and 1 is less than or equal to 8, the loop body is executed.
3. Inside the loop:
   - `c` is decremented by 1: `c = c - 1`. This means `c` becomes 0.
   - The updated value of `c` (which is now 0) is added to the variable `s`: `s = s + c`. This means `s` becomes 0.

- `c` is incremented by 2: `c = c + 2`. This means `c` becomes 2.

4. Check loop condition:
   - The condition `c <= 8` is evaluated again. Since `c` is now 2 and 2 is less than or equal to 8, the loop body is executed again.
5. Repeat steps 3 and 4:
   - The loop body is executed again, with the following changes:
     - `c` becomes 1.
     - `s` remains 0.
     - `c` becomes 3.
   - The loop body is executed again, with the following changes:
     - `c` becomes 2.
     - `s` becomes 2.
     - `c` becomes 4.
   - This process continues until `c` reaches 9. At this point, the condition `c <= 8` becomes false, and the loop terminates.
6. Exit the loop:
   - Since the loop has terminated, the code skips to the statement after the loop block.
7. Print the value of `s`:
   - The value of `s` is printed to the console.

Therefore, the output of the provided Python code is:

```
28
```

140. Output of the code:

```
val = 154
while not(val):
    val **= 2
else:
    val //= 2
print(val)
```

Output: The output will be 77. The corrected code should be:

```
val = 154
while not(val):
    val **= 2
else:
    val //= 2
print(val)
```

Now, the output will be 77.

141. Output of the code:

```
n = 10
i = 1
while i <= n:
    k = 0
    if n % i == 0:
        j = 1
        while j <= i:
            if i % j == 0:
                k = k + 1
            j = j + 1
        if k == 2:
            print(i, end=" ")
    i = i + 1
```

Output: The provided Python code defines a function that takes an integer n as input and prints all prime numbers less than or equal to n. Here's a breakdown of the code:

Initialize variables:

n is initialized to the input value (10 in this case) i is initialized to 1 Enter the while loop:

The loop iterates as long as the condition i <= n is true. Initially, i is 1 and n is 10, so the loop body is executed. Inside the loop:

Initialize a variable k to 0 Check if n is divisible by i: If n % i == 0, it means i is a factor of n. Initialize a variable j to 1 Enter an inner while loop that iterates as long as the condition j <= i is true. Inside the inner loop, check if i is divisible by j: If i % j == 0, it means j is also a factor of i. Increment the variable k by 1. Increment the variable j by 1. Exit the inner while loop. Check if the variable k is equal to 2: If k == 2, it means i has exactly two factors (1 and itself), indicating that i is a prime number. Print the value of i followed by a space. Increment i and check loop condition:

Increment the variable i by 1. Evaluate the condition i <= n. If the condition is still true, the loop body is executed again. Otherwise, the loop terminates. Exit the loop:

Since the loop has terminated, no further actions are taken. This code effectively identifies and prints all prime numbers less than or equal to the input value n.

142. Output of the code:

```
i, j = 1, 4
while True:


    if (i % 7 == 0 or j % 9 == 0):
        break
    i += 1
    j += 1
print(i, j)
```

Output: `6 9` To determine the output of the provided Python code, let's analyze the code step by step:

1. Initialize variables:
   - `i` is initialized to 1
   - `j` is initialized to 4
2. Enter the `while` loop:
   - The condition `True` is always true, so the loop body is executed repeatedly.
3. Inside the loop:
   - Check if `i` is divisible by 7 or `j` is divisible by 9:
     - If `i % 7 == 0` or `j % 9 == 0`, it means either `i` is a multiple of 7 or `j` is a multiple of 9. In this case, the `break` statement is executed, causing the loop to terminate immediately.
   - Increment `i` and `j` by 1:
     - `i` is incremented by 1, and `j` is incremented by 1.
4. Check loop condition:
   - The condition `True` is always true, so the loop body continues to execute.
5. Repeat steps 3 and 4:
   - The loop body is executed repeatedly, with `i` and `j` increasing by 1 each iteration.
6. Exit the loop:
   - At some point, either `i` will become a multiple of 7 or `j` will become a multiple of 9, triggering the `break` statement and causing the loop to terminate.
7. Print the values of `i` and `j`:
   - The final values of `i` and `j` are printed to the console.

Since the `break` statement depends on both `i` and `j` reaching specific values (multiples of 7 and 9, respectively), the exact values of `i` and `j` when the loop terminates will depend on the starting values and the increment values. However, the output will always be two integers, one a multiple of 7 and the other a multiple of 9.

143. Output of the code:

```
n = 5
c = 0
while n:
    if n > 5:
        c = c + n - 1
        n = n - 1
    else:
        c = c + n - 1
        break
print(n, c)
```

- `n` is initialized to 5, and `c` is initialized to 0.
- In the `while` loop, the condition `while n:` will be True as long as `n` is not equal to 0.
- The `if` condition `n > 5` is False since `n` is initially set to 5.

- So, it moves to the `else` block where `c = c + n - 1` is executed. `c` becomes `c + 5 - 1 = c + 4`.
- Then it encounters the `break` statement. Hence, the loop terminates.

After the loop, it prints the values of `n` and `c`.

Given that `n` has not changed since the `while` loop condition was never true (since it didn't enter the `if` block), `n` is still 5.

And `c` has been incremented by 4 in the `else` block of the loop, so `c` is 4.

Therefore, the output of the code will be:

```
5 4
```

144. Output of the program:

```
for i in range(1, 0, -1):
    print("hello")
```

Output:

In this case, the loop will execute once, starting from 1 and going down to 0. The `print("hello")` statement will execute during this single iteration. Therefore, the output of the program will be:

```
hello
```

145. Output of the program:

```
for x in range(0, 15):
    if x % 3 == 0:
        continue
    if x % 5 == 0:
        continue
    if x % 7 == 0:
        break
    print(x, end=" ")
```

Output:

```
1 2 4
```

The code uses a for loop to iterate over the numbers from 0 to 14. The range() function generates a sequence of numbers from 0 to 14, and the for loop assigns each number to the variable x.

Inside the loop, there are three if statements. The first if statement checks if x is a multiple of 3. If it is, then the continue statement is executed. This skips to the next iteration of the loop, so the current value of x is not printed.

The second if statement checks if x is a multiple of 5. If it is, then the continue statement is executed, and the current value of x is not printed.

The third if statement checks if x is a multiple of 7. If it is, then the break statement is executed. This terminates the loop, so no more numbers are printed.

Finally, the print() statement prints the current value of x, followed by a space. This is done outside of the if statements, so it only prints numbers that are not multiples of 3, 5, or 7.

146. Output of the program:

```
for i in range(1, 11):
    sum = 0
    sum += i
print(sum)
```

Output: The output will be 10 because the `sum` variable is re-initialized to 0 in each iteration of the loop, and only the last value of `i` is added.

147. Output of the program:

```
n = 6
for i in range(4, n, 1):
    if i == 5:
        break
else:
    print(n)
```

Output: The output will be nothing because the `else` block is executed only if the loop completes without encountering a `break` statement. In this case, the loop breaks when `i` is 5.

148. Output of the program:

```
n = 1
for i in range(1, n, 1):
    print("hello")
else:
    print("hi")
```

Output: The output will be "hi" because the `else` block is executed after the loop completes without encountering a `break` statement. And as the range in for loop is (1,1,1) it will complete the loop without executing print("hello") and goes directly to else statement.

149. Output of the program:

```
for i in range(1, 11):
    x = 0
    x += i
    while x < 15:
        if i % 2 == 0:
            x += 1
```

```
        else:
            x += 2
print(x)
```

Output: `15`

The program iterates over the numbers from 1 to 10. For each number, it initializes a variable x to 0 and adds the current number to it. Then, it enters a loop that continues as long as x is less than 15. Inside the loop, if the current number is even, then x is incremented by 1. Otherwise, x is incremented by 2. Finally, the value of x is printed.

The output of the program is 15 for each iteration of the loop. This is because the loop condition x < 15 is always true, and the loop body always increments x by at least 1.

150. Output of the program:

```
x = 10
if x < 15:
    print("h", end=" ")
elif x > 12:
    print("i", end=" ")
else:
    print("student", end=" ")
if x < 9:
    print("name", end=" ")
else:
    print("Name", end="")
```

- The `if` statement checks the value of `x`.
- The first condition `x < 15` is True, so it executes the corresponding `print("h", end=" ")`.
- The `elif` part `x > 12` is not evaluated because the preceding condition was True.
- The second `if` statement checks `x < 9`, which is False.
- Therefore, it executes the `else` block and prints `"Name"` without a space because the `end` parameter is not provided in this case.

Hence, the output of the program will be:

```
h Name
```

151. Output of the program:

```
x = 0
count = 0
while x < 15:
    if x % 2 == 0:
        x += 1
        continue
    if x % 3 == 0:
```

```
            x += 1
            continue
        if count == 5:
            break
        count += 1
print(x, count)
```

Output:

```
1 5
```

The program initializes two variables, x and count, to 0. Then, it enters a loop that continues as long as x is less than 15. Inside the loop, there are three if statements. The first if statement checks if x is even. If it is, then x is incremented by 1 and the loop continues. The second if statement checks if x is a multiple of 3. If it is, then x is incremented by 1 and the loop continues. The third if statement checks if count is equal to 5. If it is, then the loop is terminated. Finally, the value of x and count are printed.

The output of the program is 1 and 5. This is because the loop condition x < 15 is always true, and the loop body always increments x by at least 1. The loop is terminated when count is equal to 5, which means that the loop has been executed 5 times.

152. Output of the program:

Let's analyze the code snippet step by step:

```
x = 0
count = 0
for x in range(10):
    while x < 15:
        if x < 0:
            pass
        elif x % 2 == 0:
            x += 1
            continue
        elif x % 3 == 0:
            x += 1
            continue
        elif count == 5:
            break
        count += 1
print(x, count)
```

```
11 5
```

The program first initializes the variables `x` and `count` to 0. Then, it enters a nested loop. The outer loop iterates over the numbers from 0 to 9, and the inner loop iterates as long as `x` is less than 15. Inside the inner loop, there are four if statements. The first if statement checks if `x` is less than 0. If it is, then the loop continues without incrementing `x`. The second if statement

checks if `x` is even. If it is, then `x` is incremented by 1 and the loop continues. The third if statement checks if `x` is a multiple of 3. If it is, then `x` is incremented by 1 and the loop continues. The fourth if statement checks if `count` is equal to 5. If it is, then the loop is terminated. Finally, the loop increments `count` by 1.

After the nested loop is finished, the values of `x` and `count` are printed. In this case, `x` is equal to 11 and `count` is equal to 5.

UNIT-3

Let's break down each of the Python code snippets one by one to determine their outputs:

193.

```
def fun1(name, age=20):
    print(name, age)

fun1('Emma', 25)
```

Output: `Emma 25`

This code defines a function `fun1` that takes two parameters (`name` and `age`, with a default value of 20 for `age`) and prints them. When the function is called with `'Emma'` as `name` and `25` as `age`, it prints `Emma 25`.

194.

```
a = 10
b = 20

def change():
    global b
    a = 45
    b = 56

change()
print(a)  # Output: 10
print(b)  # Output: 56
```

Explanation: The function `change` modifies the global variable `b` to `56`. The `a` inside the function is a local variable and does not affect the global `a`, so when printed outside the function, `a` remains `10`.

195.

```
def display(b, n):
    while n > 0:
        print(b, end="")
        n = n - 1
```

```
display('z', 3)
```

Output: `zzz`

This function `display` prints the character `b` (`'z'` in this case) `n` times. So, calling `display('z', 3)` prints `'z'` three times.

196.

```
def fun(x, y, z):
    return x + y + z

print(fun(2, 30, 400))  # Output: 432
```

Explanation: The function `fun` takes three arguments and returns their sum (`2 + 30 + 400 = 432`).

197.

```
def func():
    global value
    value = "Local"

value = "Global"
func()
print(value)  # Output: Local
```

Explanation: The function `func` changes the global variable `value` to `"Local"`, which is then printed.

198.

```
def say(message, times=1):
    print(message * times)

say('Hello')        # Output: Hello
say('World', 5)     # Output: WorldWorldWorldWorldWorld
```

Explanation: The function `say` concatenates the `message` based on the `times` parameter. In the first call, `times` defaults to `1`, so it prints `'Hello'`. In the second call, `times` is explicitly set to `5`, resulting in `'World'` being printed five times.

199.

```
def sub(a, b):
    print(a - b)

sub(100, 200)  # Output: -100
sub(200, 100)  # Output: 100
```

Explanation: The function `sub` subtracts `b` from `a` and prints the result. In the first call, `100 - 200 = -100`, and in the second call, `200 - 100 = 100`.

200.

```
x = 50

def func(x):
    print('x is', x)
    x = 2
    print('Changed local x to', x)

func(x)
print('x is now', x)
```

Output:

```
x is 50
Changed local x to 2
x is now 50
```

Explanation:

- The function `func` receives a local variable `x`, initially set to the value of the global `x` (which is `50`).
- Inside the function, a new local variable `x` is assigned the value `2`, but this change only affects the local scope.
- The global `x` remains unchanged (`50`), hence the final print statement outputs the global `x` value as `50`.

201.

```
def C2F(c):
    return c * 9/5 + 32

print(C2F(100))  # Output: 212.0
print(C2F(0))    # Output: 32.0
```

Explanation:

- The function `C2F` converts Celsius to Fahrenheit using the formula `(Celsius * 9/5) + 32`.
- `C2F(100)` converts `100` Celsius to Fahrenheit (`212.0`).
- `C2F(0)` converts `0` Celsius to Fahrenheit (`32.0`).

202.

```
def function1(var1=5, var2=7):
    var2 = 9
    var1 = 3
```

```
    print(var1, "", var2)

function1(10, 12)
```

Output: `3 9`

Explanation:

- The function `function1` receives two arguments, but since default values are overridden (`10` and `12`), they won't be used.
- Inside the function, `var1` is set to `3` and `var2` is set to `9`.
- It prints these modified local variables (`3` and `9`).

203.

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'The numbers are equal'
    else:
        return y

print(maximum(2, 3))  # Output: 3
```

Explanation:

- The function `maximum` compares `x` and `y` and returns the larger number or a message if they are equal.
- In this case, `x=2` and `y=3`, so the function returns `3`.

204.

```
def power(x, y=2):
    r = 1
    for i in range(y):
        r *= x
    return r

print(power(3))    # Output: 9
print(power(3, 3)) # Output: 27
```

Explanation:

- The function `power` calculates `x` raised to the power of `y` (defaulting to `2` if `y` is not provided).
- `power(3)` calculates `3^2` and returns `9`.
- `power(3, 3)` calculates `3^3` and returns `27`.

205.

```
x = 50

def func():
    global x
    print('x is', x)
    x = 2
    print('Changed global x to', x)

func()
print('Value of x is', x)
```

Output:

```
x is 50
Changed global x to 2
Value of x is 2
```

Explanation:

- Inside the function `func`, `x` is declared as a global variable.
- It initially prints the global value of `x` (`50`), then changes the global `x` to `2`.
- After the function call, the global `x` is now `2`, which is confirmed by the final print statement.

206.

```
def add(a, b):
    return a + 5, b + 5

result = add(3, 2)
print(result)  # Output: (8, 7)
```

Explanation:

- The function `add` takes two arguments and returns a tuple containing the sum of each argument with `5`.
- Calling `add(3, 2)` returns `(8, 7)` as a tuple.

207.

```
def change(i=1, j=2):
    i = i + j
    j = j + 1
    print(i, j)

change(j=1, i=2)  # Output: 3 2
```

Explanation:

- The function `change` receives two arguments but defaults to `1` and `2` if not provided.
- Inside the function, `i` becomes `2 + 1 = 3`, and `j` becomes `1 + 1 = 2`.
- It then prints the modified values of `i` and `j`.

208.

```
def cube(x):
    return x * x * x

x = cube(3)
print(x)  # Output: 27
```

Explanation:

- The function `cube` calculates the cube of the argument.
- Calling `cube(3)` returns `27`, which is then assigned to the variable `x` and printed.

209.

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)          # Output: a is 3 and b is 7 and c is 10
func(25, c=24)      # Output: a is 25 and b is 5 and c is 24
func(c=50, a=100)   # Output: a is 100 and b is 5 and c is 50
```

Explanation:

- The function `func` has default values for `b` and `c`.
- The function is called multiple times with different argument values. If explicitly provided, arguments override default values.

210.

```
def function1(var1, var2=5):
    var1 = 2
    var3 = var1 * var2
    return var3

var1 = 3
print(function1(var1, var2))  # NameError: name 'var2' is not defined
```

Explanation:

- The code tries to print the result of `function1` with arguments `var1` and an undefined `var2`. It throws a `NameError` because `var2` is not defined in the context where it's being used.

211.

```
def printMax(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')

printMax(3, 4)  # Output: 4 is maximum
```

Explanation:

- The function `printMax` compares `a` and `b` and prints the maximum or a message if they are equal.
- In this case, `a=3` and `b=4`, so it prints `'4 is maximum'`.

212.

```
i = 0

def change(i):
    i = i + 1
    return i

change(1)
print(i)  # Output: 0
```

Explanation:

- The function `change` increments the local variable `i` by `1`, but this doesn't affect the global variable `i`, which remains `0`.

213.

```
def fun1(num):
    return num + 25

fun1(5)
print(num)  # NameError: name 'num' is not defined
```

Explanation:

- The code attempts to print `num` without defining it within the scope where it's being used, resulting in a `NameError`.

214.

What will be the last line of the output of the following Python code if test_fib(6) is called?

```
def fib(x):
    global num_fib_calls
    num_fib_calls += 1
    if x == 0 or x == 1:
        return 1
    else:
        return fib(x-1) + fib(x-2)

def test_fib(n):
    for i in range(n+1):
        global num_fib_calls
        num_fib_calls = 0
        print('fib of', i, '=', fib(i))
        print('fib called', num_fib_calls, 'times.')
```

The last line of the output is:

```
fib called 25 times.
```

This is because the function `fib(6)` is called 25 times in total. The first time it is called, it calculates the Fibonacci number of 0. The second time it is called, it calculates the Fibonacci number of 1. The third time it is called, it calculates the Fibonacci number of 2, and so on. Each time the function is called, it increments the global variable `num_fib_calls`, so by the time the function `fib(6)` is called for the 25th time, the value of `num_fib_calls` is 25.

Here is a table that summarizes the output of the program:

Input	Output
fib(0)	1
fib(1)	1
fib(2)	2
fib(3)	3
fib(4)	5
fib(5)	8
fib(6)	13

I hope this helps! Let me know if you have any other questions.

215.

```
def f(p, q, r):
    global s
    p = 10
    q = 20
    r = 30
    s = 40
    print(p, q, r, s)
```

```
p, q, r, s = 1, 2, 3, 4
f(5, 10, 15)
```

Output:

```
10 20 30 40
```

Explanation:

- The function `f` takes three arguments (`p`, `q`, `r`) and modifies them inside the function.
- `s` is declared as a global variable inside the function and assigned the value `40`.
- When `f(5, 10, 15)` is called, it prints the modified values of `p`, `q`, `r`, and `s`.

216.

If number of arguments in function definition and function call does not match, then which type of error is returned?

When the number of arguments in a function definition and function call does not match, it raises a `TypeError`.

217.

```
def power(x, y=3):
    r = 1
    for i in range(y):
        r *= x
    return r

print(power(3), end=" ")  # Output: 27
print(power(3, 3))        # Output: 27
```

Explanation:

- The `power` function calculates the power of `x` raised to `y` (defaulting to `3` if not provided).
- The first `print` statement calls `power(3)`, which calculates `3^3` and returns `27`.
- The second `print` statement explicitly calls `power(3, 3)`, which also calculates `3^3` and returns `27`.

218.

The correct statement about Python functions among the options provided is:

i) A function is a code block that only executes when called and always returns a value.

ii) A function only executes when it is called and we can reuse it in a program.

219.

```
def function1(var1=7, var2=5):
    var1 = 2
    var3 = var1 * var2
    return var3

var2 = 6
var1 = 3
print(function1(var1, var2))  # Output: 12
```

Explanation:

- The function `function1` accepts two arguments with default values.
- When called with `var1=3` and `var2=6`, it sets `var1` to `2`, calculates `var3` as `2 * 6`, and returns `12`.

220.

```
def fun(a=5, b=10, c):
    print(a**2, b//a, c**1)

fun(20, c=30)  # SyntaxError: non-default argument follows default argument
```

Explanation:

- The error arises because when defining a function, non-default arguments should not follow default arguments. In this case, `c` is a non-default argument following default arguments `a` and `b`.

221.

```
car = 20
bike = 10
cycle = 30

def new_Pur():
    global bike, cycle
    car = 30
    bike = 20
    cycle = 50

new_Pur()
print(car + 10, "", bike + 5, "", cycle + 5)  # Output: 30 25 55
```

Explanation:

- Inside `new_Pur()`, `bike` and `cycle` are declared as global variables and modified.
- After calling `new_Pur()`, the modified values of `car`, `bike`, and `cycle` are printed.

222.

```
def f():
    print(x, end=" ")
    return y

def f():
    print(y, end=" ")
    return

x = 5
y = 4
print(f())  # Output: 4 None
```

Explanation:

- The function `f()` is redefined, and the second definition overrides the first.
- Inside the redefined `f()`, `print(y, end="")` executes, printing `4`, but it returns `None` as there is no explicit return value.