Assignment -2

1. What are the two values of the Boolean data type? How do you write them?

Answer:

The two values of the Boolean data type are ‘True’ and ‘False’.
# Boolean value representing true
is_sunny = True
# Boolean value representing false
is_raining = False

2. What are the three different types of Boolean operators?

Answer:

The three different types of Boolean operators in Python are:

a. **AND (and) :** The and operator returns True if both operands are True. Otherwise, it returns False.
b. **OR (or) :** The or operator returns True if at least one of the operands is True. If both operands are False, it returns False.
c. **NOT (not) :** The not operator is a unary operator that inverts the Boolean value of its operand. If the operand is true, it returns False, and if the operand is False, it returns True.

3. Make a list of each Boolean operator’s truth tables (i.e. every possible combination of Boolean values for the operator and what it evaluate).

Answer:

a. **AND (‘and’) Truth Table**

A	B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

b. **OR (‘or’) Truth Table**

A	B	A OR B
True	True	True
True	False	True
False	True	True
False	False	False

c. **NOT (‘not’) Truth Table**

A	NOT A
True	False
False	True

4. What are the values of the following expressions?

Answer:

Expression	Explanation	Result
(5 > 4) and (3 == 5)	True and False	False
not (5 >4)	Not (True)	False
(5 > 4) or (3 == 5)	True or False	True
not ((5>4) or (3 == 5))	Not(True)	False
(True and True) and (True == False)	True and False	False
(not False) or (not True)	True or False	True

5. What are the six comparison operators?

Answer:

In Python, comparison operators are used to compare two values. Here are six common comparison operators:

- == : Equal to
- != : Not equal to
- > : Greater than
- < : Less than
- >= : Greater than or equal to
- <= : Less than or equal to

These operators return a Boolean value (True or False) based on the comparison of the operands.

6. How do you tell the difference between the equal to and assignment operators? Describe a condition and when you would use one.

Answer:

In Python, the = and == operators serve very different purposes:

Assignment Operator (=):
Purpose: Used to assign a value to a variable.
Syntax: variable = value
Example: x = 10
This assigns the value 10 to the variable x.

Equality Operator (==):
Purpose: Used to compare two values for equality.
Syntax: value1 == value2
Example:
```
if x == 10:
    print("x is equal to 10")
```
This checks if the value of x is equal to 10 and prints a message if the condition is true.

Condition and Usage
Assignment (=):
Condition: When you want to store a value in a variable for later use.
Usage Example: y = 20
This assigns the value 20 to the variable y.

Equality Comparison (==):
Condition: When you want to check if two values are the same.
Usage Example:
```
if y == 20:
    print("y is equal to 20")
```
This checks if the value of y is 20 and prints the message if the condition is true.

---

7. Identify the three blocks in this code:

Answer:

BLOCK -1
```
spam = 0
```

BLOCK-2
```
if spam == 10:
    print('eggs')
```

BLOCK-3
```
if spam > 5:
    print(‘bacon’)
else:
    print(‘ham’)

print(‘spam’)
print(‘spam’)
```

---

8. Write code that prints Hello if 1 is stored in spam, prints Howdy if 2 is stored in spam, and prints Greetings! if anything else is stored in spam.

Answer:

```
spam = input(‘Enter the value’)

if spam == 1:
    print(‘Hello’)

elif spam == 2:
   print(‘Howdy’)

else:
   print(‘Greetings!’)
```

9. If your programme is stuck in an endless loop, what keys you'll press?

Answer:

- Windows: Ctrl + C
- macOS: Control + C
- Linux: Ctrl + C

---

10. How can you tell the difference between break and continue?

Answer:

In Python, break and continue are control flow statements that are used inside loops to alter the flow of the loop. Here's how they differ:

break:

- ➢ Purpose: Immediately terminates the enclosing loop.
- ➢ Usage: When you want to exit the loop completely once a certain condition is met.
- ➢ Effect: The program control moves to the statement immediately following the loop.

Example:
```
for i in range(10):
   if i == 5:
      break
   print(i)
```
In this example, the loop will terminate when i equals 5, and the numbers 0 through 4 will be printed.

continue:

- ➢ Purpose: Skips the rest of the code inside the current iteration of the loop and proceeds to the next iteration.
- ➢ Usage: When you want to skip certain iterations of the loop but continue with the next iteration.
- ➢ Effect: The loop does not terminate but skips the remaining code for the current iteration.

Example:
```
for i in range(10):
   if i == 5:
      continue
   print(i)
```
In this example, when i equals 5, the continue statement skips the print(i) statement for that iteration. As a result, the numbers 0 through 4 and 6 through 9 will be printed, but 5 will be skipped.

Summary:
break: Ends the loop entirely.
continue: Skips to the next iteration of the loop.

11. In a for loop, what is the difference between range(10), range(0, 10), and range(0, 10, 1)?

Answer:

- **range(10)**: Implicitly starts from 0 and increments by 1 up to (but not including) 10.
- **range(0,10)**: Explicitly starts from 0 and increments by 1 up to (but not including) 10.
- **range(0,10,1)**: Explicitly starts from 0, increments by 1, and stops before 10.

All three ranges will produce the same sequence of numbers from 0 to 9. The difference lies in the explicitness of the parameters provided to the range function.

---

12. Write a short program that prints the numbers 1 to 10 using a for loop. Then write an equivalent program that prints the numbers 1 to 10 using a while loop.

Answer:

Program - 1:

```
for i in range(1,11):
    print(i)
```

Program – 2:

```
i=1
while(i<11):
   print(i)
   i=i+1
```

---

13. If you had a function named bacon() inside a module named spam, how would you call it after importing spam?

Answer:

```
spam.bacon()
```
