Assignment_2

March 12, 2023

1.What are the two values of the Boolean data type? How do you write them? The two values of the Boolean data type are ”true” and ”false”. These values represent logical truth values, where ”true” represents a condition that is true or valid, and ”false” represents a condition that is false or invalid. To assign a Boolean value to a variable in python, you can use the ”=” operator along with the desired Boolean value.

[5]:
```
# For example : A program that checks if a number is positive or negative

num = int(input("Enter a number: "))

if num >= 0:
    is_positive = True
else:
    is_positive = False

print("The number is positive:", is_positive)
```

```
Enter a number:  5
```

```
The number is positive: True
```

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

The three different types of Boolean operators are:

AND operator - denoted by ”and”. It returns True if both operands are True, otherwise it returns False.

OR operator - denoted by ”or”. It returns True if at least one of the operands is True, otherwise it returns False.

NOT operator - denoted by ”not”. It returns the opposite Boolean value of the operand. If the operand is True, then not returns False, and vice versa.

These operators are used to combine one or more Boolean values or expressions to form more complex Boolean expressions

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).

1-> AND operator (represented by and):

Operand 1 Operand 2 Result True True True True False False False True False False False False2-> OR operator (represented by or):

Operand 1 Operand 2 Result True True True True False True False True True False False False3-> NOT operator (represented by not):

Operand Result True False False True4. What are the values of the following expressions?

(5 > 4) and (3 == 5) = False not (5 > 4) = False (5 > 4) or (3 == 5) = True not ((5 > 4) or (3 == 5)) = False (True and True) and (True == False) = False (not False) or (not True) = True

5. What are the six comparison operators?

The six comparison operators are:

1. Equal to (==) - Returns True if the operands are equal, otherwise returns False.
2. Not equal to (!=) - Returns True if the operands are not equal, otherwise returns False.
3. Greater than (>) - Returns True if the left operand is greater than the right operand, otherwise returns False.
4. Less than (<) - Returns True if the left operand is less than the right operand, otherwise returns False.
5. Greater than or equal to (>=) - Returns True if the left operand is greater than or equal to the right operand, otherwise returns False.
6. Less than or equal to (<=) - Returns True if the left operand is less than or equal to the right operand, otherwise returns False.

[12]:
```
# Equal to (==)
x = 5
y = 5
if x == y:
    print("x is equal to y")

# Not equal to (!=)
x = 5
y = 10
if x != y:
    print("x is not equal to y")

# Greater than (>)
x = 10
y = 5
if x > y:
    print("x is greater than y")

# Less than (<)
x = 5
y = 10
if x < y:
```

```
    print("x is less than y")

# Greater than or equal to (>=)
x = 10
y = 10
if x >= y:
    print("x is greater than or equal to y")

# Less than or equal to (<=)
x = 5
y = 10
if x <= y:
    print("x is less than or equal to y")
```

```
x is equal to y
x is not equal to y
x is greater than y
x is less than y
x is greater than or equal to y
x is less than or equal to y
```

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

The equal to operator (==) and the assignment operator (=) are two different operators used in Python.

The equal to operator (==) is used to compare two values to see if they are equal, and it returns a Boolean value of True or False. For example, x == y compares the value of x to the value of y, and returns True if they are equal and False otherwise.

On the other hand, the assignment operator (=) is used to assign a value to a variable. For example, x = 5 assigns the value 5 to the variable x.

[14]:
```
x = 5
y = 10

if x == y:
    print("x is equal to y")
else:
    print("x is not equal to y")

x = y

print("The value of x is now:", x)
```

```
x is not equal to y
The value of x is now: 10
```

7. Identify the three blocks in this code: spam = 0 if spam == 10: print('eggs') if spam > 5:

print('bacon') else: print('ham') print('spam') print('spam')

[ ]:
```
#First block
spam = 0

#Second Block
if spam == 10:
    print('eggs')


#Third Block
if spam > 5:
    print('bacon')
else:
    print('ham')
    print('spam')
    print('spam')
```

```
ham
spam
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.

[18]:
```
spam = 3

if spam == 1:
    print("Hello")
elif spam == 2:
    print("Howdy")
else:
    print("Greetings!")
```

```
Greetings!
```

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

If I stuck in an endless loop, I usually stop it by pressing CTRL + C on keyboard.

This keyboard shortcut sends an interrupt signal to the program, which causes it to immediately stop running. It's a useful way to force a program to exit when it's not responding or when it's stuck in an infinite loop.

If CTRL + C doesn't work or I am unable to use it, I may need to force quit the program using my operating system's task manager or equivalent tool.

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

"break" and "continue" are two control flow statements in Python that are used to modify the behavior of loops.

[20]:
```
#"break" is used to exit a loop early if a certain condition is met. When break␣
 ↪is encountered inside a loop, the loop is immediately terminated and the␣
 ↪program continues with the code that follows the loop.
for i in range(1, 10):
    if i == 5:
        break
    print(i)
#In this code, we use a for loop to iterate over the numbers from 1 to 9.␣
 ↪Inside the loop, we check if i is equal to 5. If it is, we use break to exit␣
 ↪the loop early. As a result, the program only prints the numbers from 1 to 4.
```

```
1
2
3
4
```

[21]:
```
#continue, on the other hand, is used to skip over an iteration of a loop if a␣
 ↪certain condition is met. When continue is encountered inside a loop, the␣
 ↪current iteration of the loop is immediately terminated, and the program␣
 ↪continues with the next iteration of the loop.
for i in range(1, 10):
    if i == 5:
        continue
    print(i)
#In this code, we use a for loop to iterate over the numbers from 1 to 9.␣
 ↪Inside the loop, we check if i is equal to 5. If it is, we use continue to␣
 ↪skip over that iteration of the loop. As a result, the program prints all␣
 ↪the numbers from 1 to 9 except for 5.

#In summary, break is used to exit a loop early, while continue is used to skip␣
 ↪over an iteration of a loop.
```

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

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

In a for loop, range(10), range(0, 10), and range(0, 10, 1) all produce the same sequence of integers, which is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. However, there is a difference in the way these three expressions generate the sequence.

[22]:
```
#range(10) is a built-in function in Python that generates a sequence of␣
 ↪integers from 0 up to, but not including, the number 10. It is equivalent to␣
 ↪range(0, 10, 1) and range(0, 10).
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

[23]:
```
#range(0, 10) is also a built-in function in Python that generates a sequence␣
 ↪of integers from 0 up to, but not including, the number 10. It is equivalent␣
 ↪to range(10) and range(0, 10, 1).
for i in range(0, 10):
    print(i)
#This will output the same numbers as the previous example, from 0 to 9,␣
 ↪inclusive.
```

```
0
1
2
3
4
5
6
7
8
9
```

[24]:
```
#range(0, 10, 1) is also a built-in function in Python that generates a␣
 ↪sequence of integers from 0 up to, but not including, the number 10, with a␣
 ↪step size of 1. It is equivalent to range(0, 10) and range(10)
for i in range(0, 10, 1):
    print(i)
#This will output the same numbers as the previous examples, from 0 to 9,␣
 ↪inclusive.
```

```
0
1
2
3
```

```
4
5
6
7
8
9
```

In summary, range(10), range(0, 10), and range(0, 10, 1) all generate the same sequence of integers, but the second and third forms are more explicit about the starting and stepping values.

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.

[25]:
```
#for loop

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

```
1
2
3
4
5
6
7
8
9
10
```

[26]:
```
#while loop
i = 1
while i <= 10:
    print(i)
    i += 1
```

```
1
2
3
4
5
6
7
8
9
10
```

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

If we had a function named bacon() inside a module named spam, we would call it after importing

spam using dot notation as follows:

import spam spam.bacon()

[ ]: