Python Conditional Statements

S1 MCA

Conditional Statements

- Conditional statements in Python allow for different blocks of code to be executed based on whether certain conditions are true or false. These statements are crucial for controlling the flow of a program and enabling decision-making.
- if
- if-else
- if-elif-else (or if-elif ladder)

- **If Statement**

A Python if statement is a fundamental control flow statement used for decision-making. It allows a specific block of code to be executed only if a given condition evaluates to True.

Syntax:

```
if condition:
        # Code to execute if the condition is True
        statement_1
        statement_2
         # ...


age = 18
if age >= 18:
        print("You are eligible to vote.")
```

- **Elif**

The **elif** keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

→ In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

- **else**

The **else** keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

- You can also have an else without the elif

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

- **Short Hand If**

```
if a > b: print("a is greater than b")
```

Ternary Operators, or Conditional Expressions.

- **Short Hand If ... Else**

status = "Adult" if age >= 18 else "Minor"

- **And**

The and keyword is a logical operator, and is used to combine conditional statements:

Test if `a` is greater than `b`, AND if `c` is greater than `a`:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Test if `a` is greater than `b`, OR if `a` is greater than `c`:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Not

The not keyword is a logical operator, and is used to reverse the result of the conditional statement:

Test if `a` is NOT greater than `b`:

```
a = 33
b = 200
if not a > b:
  print("a is NOT greater than b")
```

- **Nested If**

You can have if statements inside if statements, this is called nested if statements.

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

→Output:

```
Above ten,
and also above 20!
```

- **The pass Statement**

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200

if b > a:
  pass


# having an empty if statement like this, would raise an error without the pass statement
```

Python Match

- Instead of switch(expression) we use Match in Python
- The match statement is used to perform different actions based on different conditions.
- Instead of writing many if..else statements, you can use the match statement.
- The match statement selects one of many code blocks to be executed.

- Syntax:

match *expression*:
  case x:
    code block
  case y:
    code block
  case z:
    code block

This is how it works: The match expression is evaluated once.

The value of the expression is compared with the values of each case.

If there is a match, the associated block of code is executed.

```
day = 4
match day:
  case 1:
    print("Monday")
  case 2:
    print("Tuesday")
  case 3:
    print("Wednesday")
  case 4:
    print("Thursday")
  case 5:
    print("Friday")
  case 6:
    print("Saturday")
  case 7:
    print("Sunday")
```

- **Default Value**

Use the underscore character _ as the last case value if you want a code block to execute when there are not other matches:

```
day = 4
match day:
  case 6:
    print("Today is Saturday")
  case 7:
    print("Today is Sunday")
  case _:
    print("Looking forward to the Weekend")
```

→ Output : Looking forward to the Weekend

- **Combine Values**

Use the pipe character | as an or operator in the case evaluation to check for more than one value match in one case:

```
day = 4
match day:
  case 1 | 2 | 3 | 4 | 5:
    print("Today is a weekday")
  case 6 | 7:
    print("I love weekends!")
```

→ Output : Today is a weekday

- **If Statements as Guards**

You can add if statements in the case evaluation as an extra condition

```
month = 5
day = 4
match day:
  case 1 | 2 | 3 | 4 | 5 if month == 4:
    print("A weekday in April")
  case 1 | 2 | 3 | 4 | 5 if month == 5:
    print("A weekday in May")
  case _:
    print("No match")
```

→ Output : A weekday in May

Q

Print "1" if `a` is equal to `b`, print "2" if `a` is greater than `b`, otherwise print "3".

```
a = 50
b = 10
[redacted] a [redacted] b
  print("1")
[redacted] a [redacted] b
  print("2")
[redacted]
  print("3")
```

A

```
a = 50
b = 10
if a == b:
  print("1")
elif a > b:
  print("2")
else:
  print("3")
```