Conditions in python

Lesson Plan

PW SKILLS

Today's checklist:

Intro to conditions
If,else and elif Conditions
Truly Value & Fasly Value
Nested Conditions
Ternary Operator
Match and Case in Python

Conditional Statements:

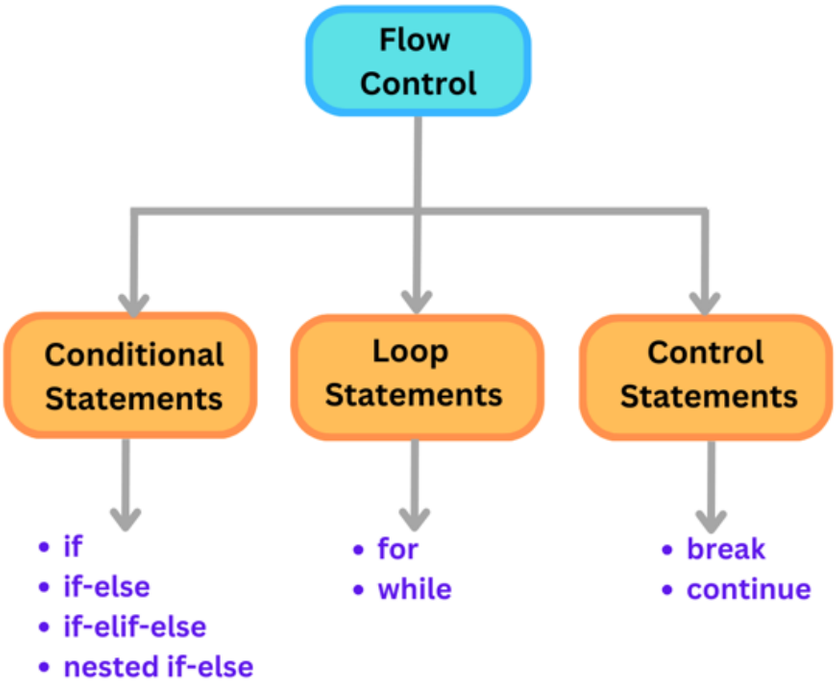
- Conditional statements, often known as control structures, are an essential component of programming. They enable you to make code decisions based on predefined conditions.
- The primary conditional statements in Python are if, elif (short for "else if"), and else.

Purpose of Conditional Statements:

The purpose of conditional statements in programming is to enable the execution of different code blocks based on specific conditions. Here are four key purposes of conditional statements:

1. **Decision Making:** Conditional statements allow a program to make decisions by evaluating conditions. They help determine which set of instructions to execute depending on whether the conditions are met or not.
2. **Control Flow:** Conditional statements control the flow of a program, ensuring that the code follows a specific path or branches based on the given conditions. This helps create responsive and adaptable programs.
3. **Error Handling:** Conditional statements are used for error handling. They can identify and respond to exceptional situations, preventing program crashes or unexpected behavior by executing specific error-handling code.
4. **Customization:** Conditional statements provide the means to customize the behavior of a program for different scenarios or inputs. They allow programmers to tailor the program's response to various user interactions or data inputs, making the software more versatile and user-friendly.

Python Conditions and If statements



Python supports the usual logical conditions from mathematics:

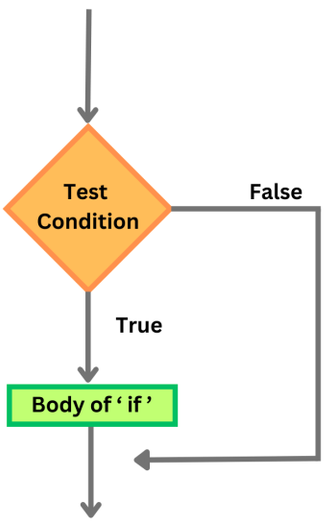
- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:



a = 33
b = 200

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

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example
If statement, without indentation (will raise an error):

a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error

Elif

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

Example
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.

Example
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:

Example
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 you have only one statement to execute, you can put it on the same line as the if statement.

Example
One line if statement:

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

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example
One line if else statement:

```
a = 2
b = 330

print("A") if a > b else print("B")
```

This technique is known as Ternary Operators, or Conditional Expressions.

You can also have multiple else statements on the same line:

Example
One line if else statement, with 3 conditions:

```
a = 330
b = 330

print("A") if a > b else print("=") if a == b else print("B")
```

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

Example
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:

Example
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:

Example
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.

Example

```
x = 41

if x > 10:
print("Above ten,")

if x > 20:
print("and also above 20!")

else:
print("but not above 20.")
```

Python Ternary Operator

The Python ternary operator determines if a condition is true or false and then returns the appropriate value in accordance with the result. The ternary operator is useful in cases where we need to assign a value to a variable based on a simple condition, and we want to keep our code more concise — all in just one line of code. It's particularly handy when you want to avoid writing multiple lines for a simple if-else situation.

Syntax of Ternary Operator in Python:

Syntax: [on_true] if [expression] else [on_false]

expression: conditional_expression | lambda_expr

Simple Method to Use Ternary Operator

In this example, we are comparing and finding the minimum number by using the ternary operator. The expression min is used to print a or b based on the given condition. For example, if a is less than b then the output is a, if a is not less than b then the output is b.

```
# Program to demonstrate conditional operator
a, b = 10, 20

# Copy value of a in min if a < b else copy b
min = a if a < b else b

print(min)
```

Output
10

What is a 'match-case' statement?

We have to first pass a parameter then try to check with which case the parameter is getting satisfied. If we find a match we will do something and if there is no match at all we will do something else.

Pseudocode for Python match-case Statement

The match statement is initialized with the match keyword creating a block and taking a parameter (here the name is also a parameter) and then steps down to the various cases using the case keyword and the pattern, for the pattern to match the parameter. The " _ " is the wildcard character which is run when nothing is matched.

```
parameter = "physicswallah"

match parameter:

  case first :
     do_something(first)

  case second :
      do_something(second)

  case third :
     do_something(third)
     ..............
     ..............

  case n :
     do_something(n)
   case _ :
      nothing_matched_function()
```

Truthy vs Falsy Values

In Python, individual values can evaluate to either True or False.

The Basis rules are:

Values that evaluate to False are considered Falsy.
Values that evaluate to True are considered Truthy.

Falsy Values Includes:

1) Sequences and Collections:

Empty lists []
Empty tuples ()
Empty dictionaries {}
Empty sets set()
Empty strings " "
Empty ranges range(0)

2) Numbers: Zero of any numeric type.

Integer: 0
Float: 0.0
Complex: 0j

3) Constants:

None
False
Falsy values were the reason why there was no output in our initial example when the value of the number was zero.

Truthy Values Includes:

Non-empty sequences or collections (lists, tuples, strings, dictionaries, sets).
Numeric values that are not zero.

Constant: True

This is why the value of a printed in our initial example because its value of a number was 7(a truthy value):

Built-in bool() function

You can check if a value is either truthy or falsy with the built-in bool() function. This function is used to return or convert a value to a Boolean value i.e., True or False, using the standard truth testing procedure

Syntax: bool(parameter)

You only need to pass the value as an argument.

Example:

```
bool(7)
# True

bool(0)
 #False

bool([])
# False

bool({7,4})
#True

bool(-4)
# True

bool(0.0)
# False
```

```
bool(None)
# False

bool(1)
#True

bool(range(0))
# False

bool(set())
# False

bool([1,2,3,4])
# True
```

Output:

Now let's see a program for better understanding of Truthy and Falsy value.

Example:

```
# define a function for checking
# number is even or odd
def even_odd(number):

  if number % 2:

    # since num % 2 is equal to 1
    # which is Truthy Value
    return 'odd number'

  else:

    # since num%2 is equal to 0
    # which is Falsy Value.
    return 'even number'

result1 = even_odd(7)

# prints odd
print(result1)

result2 = even_odd(4)

# prints even
print(result2)
```

Output:

```
odd number
even number
```