python.epsilonpi.club

Conditionals, loops and iterations

VISWANATH AKHIL

Comparison Operators

<

Less than

<=

Less than or Equal to

==

Equal to

>=

Greater than or Equal to

>

Greater than

!=

Not equal

Condtionals

- Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false
- These conditions can be used in several ways, most commonly in "if statements" and loops.
- Examples are if , else ,elseif, switch(doesn't work in python)

Python 'if' statement

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

```
a = 33
b = 200
if b > a:
print("b is greater than a")
```

Python 'else' statement

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")
```

Try it Yourself »

Python 'elseif' statement

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

```
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")
```

Try it Yourself »

Python nested 'if' statement

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.")
```

Shorthand 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

```
a = 2
b = 330
print("A") if a > b else print("B")
```

Logical Operators

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

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

Conditionals

Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false

Loops and Iterations

repetitive control structures are a way for computer programs to repeat one or more various steps depending on conditions set either by the programmer initially or real-time by the actual program

while loop in Python

With the **while** loop we can execute a set of statements as long as a condition is true

```
i = 1
while i < 6:
  print(i)
  i += 1
```

while with else in Python

With the else statement we can run a block of code once when the condition no longer is true

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

'break' & 'continue'

With the break statement we can stop the loop even if the while condition is true

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Try it Yourself »

With the continue statement we can stop the current iteration, and continue with the next

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

'for' loop in python

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string)

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

nested 'for' loop in python

A nested loop is a loop inside a loop.
The "inner loop" will be executed one time for each iteration of the "outer loop"

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```