PYTHON TUTORIAL FOR BEGINNERS

Source: www.youtube.com/@RishabhMishraOfficial

Chapter - 14

Loops in Python

- Loops & Types
- While Loop
- For Loop
- Range Function
- Loop Control Statements

Loops in Python

Loops enable you to perform **repetitive tasks** efficiently without writing redundant code. They iterate over a sequence (like a list, tuple, string, or range) or execute a block of code as long as a specific **condition is met**.

Types of Loops in Python

1. While loop
2. For loop
3. Nested loop

While Loop

The while loop **repeatedly** executes a block of code as long as a given condition remains `True`. It checks the condition before each iteration.

Syntax:

```
while condition:
    # Code block to execute
```

Example:

```
Print numbers from 0 to 3
count = 0
while count < 4:        # Condition
    print(count)
    count += 1
# Output: 0  1  2  3
```

While Loop Example

`else Statement:` An else clause can be added to loops. It executes after the loop finishes normally (i.e., not terminated by break). ***Example:***

```
count = 3
while count > 0:        # Condition
    print("Countdown:", count)
    count -= 1
else:
    print("Liftoff!")  # Run after while loop ends
```

For Loop

The `for` loop in Python is used to iterate over a sequence (such as a list, tuple, dictionary, set, or string) and execute a block of code for **each element** in that sequence.

Syntax:

```
for variable in sequence:
    # Code block to execute
```

Example: *iterate over each character in language*

```
language = 'Python'
for x in language:
    print(x)        # Output: P  y  t  h  o  n
```

Using range() Function

To **repeat** a block of code a specified number of times, we use the `range()` function.

The `range()` function returns a sequence of numbers, starting from 0 by default, increments by 1 (by default), and stops before a specified number.

Syntax:

```
range(stop)
range(start, stop)
range(start, stop, step)
```

- `start`: (optional) The beginning of the sequence. Defaults is 0. (inclusive)
- `stop`: The end of the sequence (exclusive).
- `step`: (optional) The difference between each number in the sequence.

  Defaults is 1.

range() Function Example

Example1: *Basic usage with One Argument - Stop*

```
for i in range(5):
    print(i)
# Output: 0  1  2  3  4
```

Example2: *Basic usage with Start, Stop and Step*

```
for i in range(1, 10, 2):
    print(i)
# Output: 1  3  5  7  9
```

For Loop Example

`else Statement:` An else clause can be added to loops. It executes after the loop finishes normally (i.e., not terminated by break).

Example:

```
for i in range(3):
    print(i)
else:
    print("Loop completed")
# Output: 0  1  2  Loop Completed
```

while loop VS for loop

`while loop`

- A while loop keeps running as long as a **condition is true**.
- It is generally used when you **don't know** how many iterations will be needed beforehand, and loop continues based on a condition.

`for loop`

- A for loop **iterates over a sequence** (like a strings, list, tuple, or range) and runs the loop for each item in that sequence.
- It is used when **you know** in advance how many times you want to repeat a block of code.

Loop Control Statements

Loop control statements allow you to **alter** the normal flow of a loop.
Python supports 3 clauses within loops:

- `pass statement`
- `break Statement`
- `continue Statement`

Loop Control - pass Statement

`pass Statement:` The pass statement is used as a **placeholder** (**it does nothing**) for the future code, and runs entire code without causing any syntax error. *(already covered in functions)*

Example:

```
for i in range(5):
    # code to be updated
      pass
```

Above example, the loop **executes** without error using **pass** statement

Loop Control - break Statement

`break Statement:` The break statement terminates the loop entirely, exiting from it immediately.

Example:

```
for i in range(5):
   if i == 3:
       break
   print(i)    # Output: 0  1  2
```

Above example, the loop **terminated** when condition met true for i == 3

Loop Control - continue Statement

`continue Statement:` The continue statement **skips** the current iteration and moves to the next one.

Example:

```
for i in range(5):
    if i == 3:
        continue
    print(i)      # Output: 0  1  2  4
```

Above example, the loop **skips** when condition met true for i == 3

break vs continue Statement

break Statement example

```
# pass statement
count = 5
while count > 0:
    if count == 3:
        pass
    else:
        print(count)
    count -= 1

# Output: 5  4  2  1
```

continue Statement example

```
# continue statement: don't try - infinite loop
count = 5
while count > 0:
    if count == 3:
        # continue
    else:
        print(count)
    count -= 1

# Output: 5  4  3  3…….
```

Validate User Input

```
# validate user input: controlled infinite while loop using break statement

while True:
    user_input = input("Enter 'exit' to STOP: ")
    if user_input == 'exit':
        print("congarts! You guessed it right!")
        break
    print("sorry, you entered: ", user_input)
```

Python Tutorial Playlist: **Click Here**

https://www.youtube.com/playlist?list=PLdOKnrf8EcP384Ilxra4UlK9BDJGwawg9