

Basic Programming with Python

Prepared By:

Professor Dr. Md. Mijanur Rahman

Department of Computer Science & Engineering

Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

www.mijanrahman.com

3

Control Structures in Python

CONTENTS

3.3 Loop Control Statements	1
3.3.1. Break Statement	2
3.3.2. Continue Statement	4
3.3.3. Pass Statement	6

3.3 LOOP CONTROL STATEMENTS

Loop control statements in Python are special commands that allow us to alter the flow of loops. They enable us to control the iteration process and decide when to continue, break, or skip certain iterations within loops. Python provides three main loop control statements:

- 1. Break Statement:** The break statement is used to exit the loop prematurely. When a break statement is encountered inside a loop, the loop is terminated immediately, and program control resumes at the next statement following the loop.

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

3. Control Structures in Python

In this example, the loop terminates when *i* becomes 3, and the program control moves to the next statement after the loop.

- 2. Continue Statement:** The continue statement is used to skip the rest of the code inside the loop for the current iteration and proceed to the next iteration of the loop.

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

In this example, the loop skips printing the number 2 but continues with the next iteration to print the remaining numbers.

- 3. Pass Statement:** The pass statement is a null operation that does nothing. It is used when a statement is required syntactically but you do not want to execute any code.

```
for i in range(5):
    if i == 2:
        pass
    else:
        print(i)
```

In this example, the pass statement is used to do nothing when *i* is 2, and the loop continues executing for other values of *i*.

Loop control statements provide flexibility and control over loop execution, allowing us to handle various scenarios and conditions within loops. They are commonly used in conjunction with conditional statements to control the flow of the program based on specific conditions.

3.3.1. Break Statement

Python break statement is used to terminate the current loop and resumes execution at the next statement, just like the traditional break statement in C. When encountered within a loop (such as for or while), the break statement causes the loop to terminate immediately, and program execution resumes at the next statement following the loop.

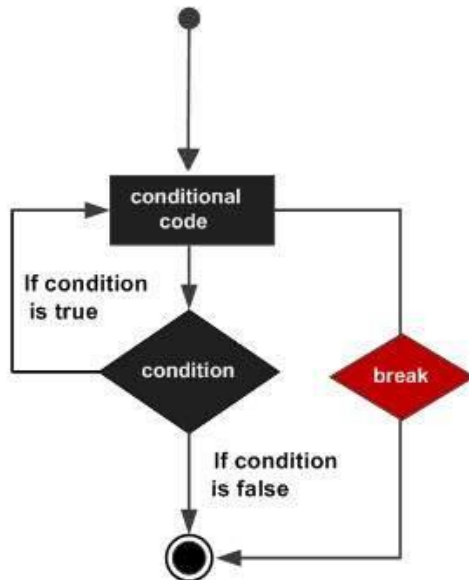
The break statement is typically used within loops to exit the loop based on a specific condition, rather than completing all iterations of the loop. It is useful when we want to terminate a loop early based on certain criteria, without needing to complete the remaining iterations.

The general syntax of the break statement is as follows:

```
for item in iterable:
    if condition:
        break
or
while condition:
    if condition:
        break
```

3. Control Structures in Python

The break statement is usually placed within an if statement inside the loop. When the condition specified with the break statement evaluates to True, the loop is terminated immediately, and program control moves to the next statement outside the loop. If the break statement is not encountered, the loop continues to iterate until its completion. The following flowchart illustrates how a break statement works in a program:



An example of a break statement is given below:

```
# Example using a for loop
for i in range(5):
    print(i)
    if i == 2:
        break
```

In this example, the for loop iterates over the numbers from 0 to 4. Inside the loop, each number *i* is printed. When *i* becomes 2, the break statement is encountered, causing the loop to terminate immediately. As a result, the loop exits after printing 0, 1, and 2, and program control moves to the next statement after the loop.

Example 3.18: Checking for prime number using break statement in Python.

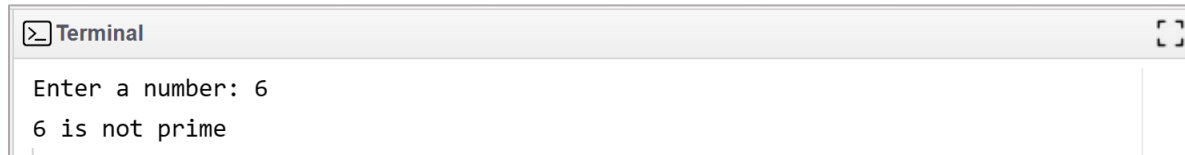
```
num = int(input("Enter a number: "))

for x in range(2,num):
    if num%x == 0:
        print("{} is not prime".format(num))
        break
    else:
        print("{} is prime".format(num))
```

In this script, we use a for loop over numbers from 2 to the desired number-1. If it divisible by any value of looping variable, the number is not prime, hence the program breaks from the loop. If the

number is not divisible by any number between 2 and x-1, the else block prints the message that the given number is prime.

Output:



```
Terminal
Enter a number: 6
6 is not prime
```

3.3.2. Continue Statement

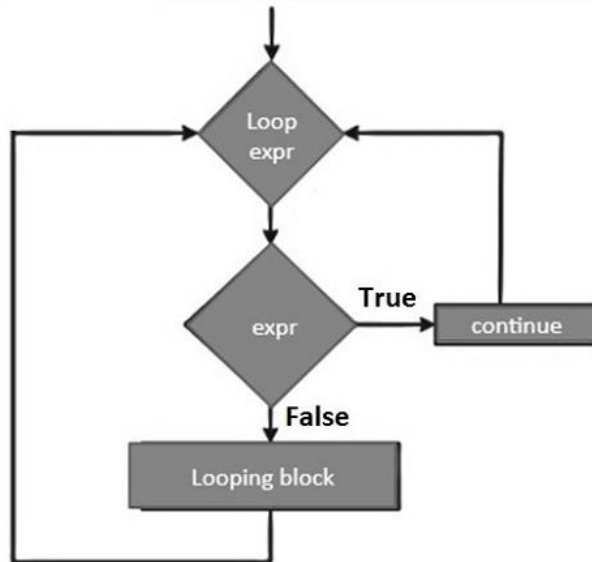
In Python, the continue statement is used inside loops to skip the rest of the code inside the loop for the current iteration and proceed to the next iteration of the loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration. Essentially, it allows us to bypass the remaining code inside the loop block and move to the next iteration of the loop. The continue statement is typically used within loops in conjunction with conditional statements to selectively skip certain iterations based on specific conditions.

The continue statement can be used in both while and for loops. The general syntax of the continue statement is as follows:

```
for item in iterable:
    if condition:
        continue
    # Code here will be skipped for certain iterations
or
while condition:
    if condition:
        continue
    # Code here will be skipped for certain iterations
```

The continue statement is usually placed within an if statement inside the loop. When the condition specified with the continue statement evaluates to True, the rest of the code inside the loop block for the current iteration is skipped, and program control moves to the next iteration of the loop. If the condition is not met, the loop continues normally with the execution of the remaining code inside the loop block.

The flowchart of the continue statement looks like:



The continue statement is just the opposite to that of break. It skips the remaining statements in the current loop and starts the next iteration. For an example:

```
# Example using a for loop
for i in range(5):
    if i == 2:
        continue
    print(i)
```

In this example, the for loop iterates over the numbers from 0 to 4. Inside the loop, there's an if statement that checks if i is equal to 2. When i is 2, the continue statement is encountered, causing the rest of the code inside the loop block for that iteration (in this case, print(i)) to be skipped. As a result, 2 is not printed, and program control moves to the next iteration of the loop.

Example 3.19: Checking prime factors using continue statement in Python.

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

print ("Prime factors for ", num, "are:")
d = 2
while num > 1:
    if num%d==0:
        print (d)
        num=num/d
        continue
    d=d+1
```

In this example, we use a continue statement to find the prime factors of a given number. To find prime factors, we successively divide the given number starting with 2 using while loop, increment the divisor and continue the same process till the input reduces to 1.

Output:

```
Terminal
Enter a number: 60
Prime factors for 60 are:
2
2
3
5
```

3.3.3. Pass Statement

In Python, the pass statement is a null operation that does nothing when executed. It acts as a placeholder for syntactical correctness and is typically used when a statement is required by Python syntax, but no action needs to be taken. The pass statement is often used as a placeholder for future code or to create empty code blocks.

The general syntax of the pass statement is as follows:

```
if condition:
    pass
```

The pass statement is simply the keyword pass by itself. It can be used anywhere in Python where a statement is syntactically required. When Python encounters a pass statement, it does nothing and continues with the execution of the next statement.

The pass statement is commonly used when we want to define a code block that will be filled in later or when we want to create a placeholder for future code. It is also useful in defining empty function or class definitions.

Example uses of the pass statement:

- Placeholder in an if statement:

```
if condition:
    pass          # Placeholder for future code
else:
    Code-block
```

- Empty function definition:

```
def my_function():
    pass          # Empty function body
```

- Placeholder in a class definition:

```
class MyClass:
    def my_method(self):
        pass      # Placeholder for method implementation
```

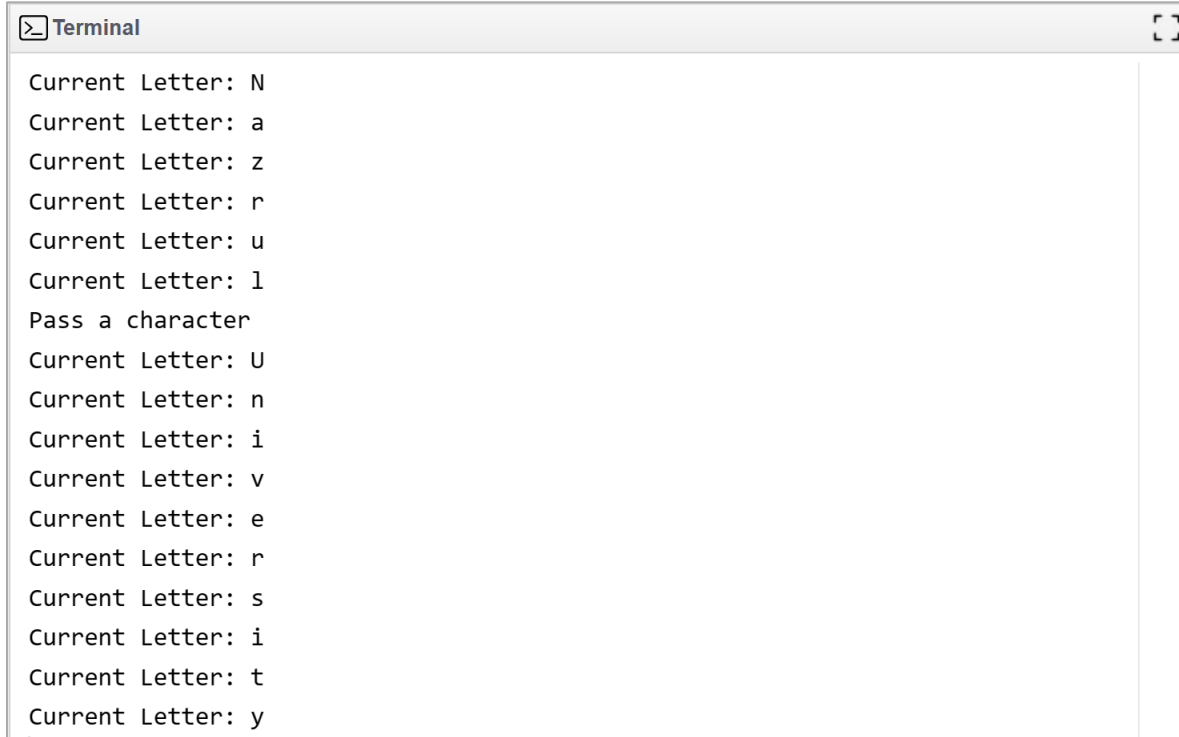
Example 3.20: Illustration of the usage of pass statement in Python.

```
for letter in 'Nazrul-University':
    if letter == '-':
```

3. Control Structures in Python

```
pass
print ('Pass a character')
else:
print ('Current Letter:', letter)
```

Output:



```
Terminal
Current Letter: N
Current Letter: a
Current Letter: z
Current Letter: r
Current Letter: u
Current Letter: l
Pass a character
Current Letter: U
Current Letter: n
Current Letter: i
Current Letter: v
Current Letter: e
Current Letter: r
Current Letter: s
Current Letter: i
Current Letter: t
Current Letter: y
```
