Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 5

Loops

Course Roadmap

Part 1: Python Basics and Functions

Lecture 1: Python Overview

Lecture 2: Variable Types

Lecture 3: Basic Operations

Lecture 4: Conditions

Lecture 5: Loops

Lecture 6: Functions

© Prepared by: Mohamed Ayman

Lecture Agenda

We will discuss in this lecture the following topics

1- Loop Definition

2- While Loop Statements

3- Else with While Loop

4- For Loop Statements

5- Else with For Loop

6- Nested Loops

7- Loop Control Statements


Let's
START UP

Lecture Agenda

Section 1: Loop Definition

Section 2: While Loop Statements

Section 3: Else with While Loop

Section 4: For Loop Statements

Section 5: Else with For Loop

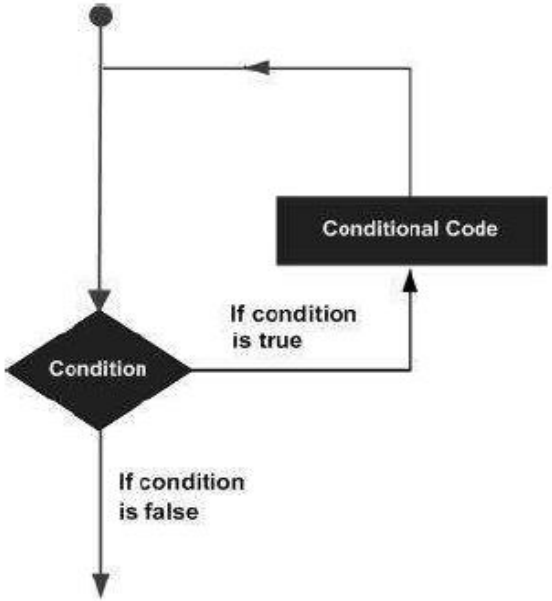
Section 6: Nested Loops

Section 7: Loop Control Statements

© Prepared by: Mohamed Ayman

Loop Definition

- In general, statement are executed sequentially - The first statement in a function is executed first, followed by the second, and so on.
- There may be a situation when you need to execute a block of code several number of times.
- Programming language provide various control structure that allow more complicated execution paths.
- A loop statement allows us to execute a statement of group of statement multiple times.



© Prepared by: Mohamed Ayman

Types of Loops

- Python programming language provides the following types of loops to handle looping requirement.
- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true, We generally use this loop when we don't know beforehand, the number of times to iterate.
- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, or for loop.

© Prepared by: Mohamed Ayman

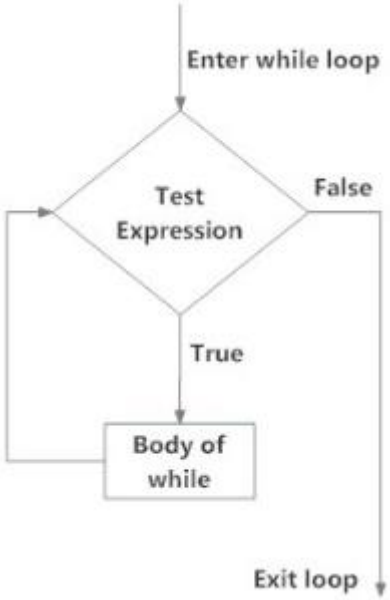
Lecture Agenda

- ✓ Section 1: Loop Definition
- **Section 2: While Loop Statements**
- Section 3: Else with While Loop
- Section 4: For Loop Statements
- Section 5: Else with For Loop
- Section 6: Nested Loops
- Section 7: Loop Control Statements

While Loop Statements

- A while loop statement in Python programming language repeatedly executes a target statement as long as given condition is true.

```
while expression:
    statement(s)
```



- Expression can be TRUE by any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.
- In while loop, test expression is checked first. The body of the loop is entered only if the expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the expression evaluates to False.
- In Python, the body of the while loop is determined through indentation, Body starts with indentation and the first unindented line marks the end.
- Python interprets any non-zero value as True. None and 0 are interpreted as False.

© Prepared by: Mohamed Ayman

While Loop Statements

Example:

```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

Output:

```
0
1
2
the program completed
```

© Prepared by: Mohamed Ayman

While Loop Statements

Iterations Trace:

1-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

2-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

3-
```
i = 0
while i < 3:
    print(i)                    0
    i += 1
print('the program completed')
```

4-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

While Loop Statements

Iterations Trace:

1-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

2-
```
i = 0
while i < 3:
    print(i)                    1
    i += 1
print('the program completed')
```

3-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

4-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

While Loop Statements

Iterations Trace:

1-
```
i = 0
while i < 3:
    print(i)                    2
    i += 1
print('the program completed')
```

2-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

3-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

4-
```
i = 0
while i < 3:
    print(i)
    i += 1
print('the program completed')
```

Lecture Agenda

- ✔ Section 1: Loop Definition
- ✔ Section 2: While Loop Statements
- **Section 3: Else with While Loop**
- Section 4: For Loop Statements
- Section 5: Else with For Loop
- Section 6: Nested Loops
- Section 7: Loop Control Statements

Else with While Loop

- Python support having an else statement associated with a loop statement.
- If the else statement is used with a while loop, the else statement is executed when the condition becomes false.
- The else part is executed if the condition in the while loop evaluates to False. The while loop can be terminated with a break statement.
- The while loop can be terminated with a break statement. In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

```
while expression:
    statement(s)
else:
    statement(s)
```

Else with While Loop

Example:

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

Output:

```
0
1
2
i not less than 3
```

© Prepared by: Mohamed Ayman

Else with While Loop

Iterations Trace:

1-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

2-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

3-
```
i = 0
while i < 3:
    print(i)                    0
    i += 1
else:
    print('i not less than 3')
```

4-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

© Prepared by: Mohamed Ayman

Else with While Loop

Iterations Trace:

1-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

2-
```
i = 0
while i < 3:
    print(i)                    1
    i += 1
else:
    print('i not less than 3')
```

3-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

Else with While Loop

Iterations Trace:

1-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

2-
```
i = 0
while i < 3:
    print(i)                    2
    i += 1
else:
    print('i not less than 3')
```

3-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

Else with While Loop

Iterations Trace:

1-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

2-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

3-
```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('i not less than 3')
```

© Prepared by: Mohamed Ayman

Quiz

- What is the output of the following code?

```
i, sum = 0, 0
while i <= 4:
    sum += i
    i += 1
print(sum)
```

- ☐ 0
- ☐ 10
- ☐ 4
- ☐ none of the above

```
i, sum = 0, 0
while i >= 4:
    sum += i
    i += 1
print(sum)
```

- ☐ 0
- ☐ 10
- ☐ 4
- ☐ none of the above

Quiz Solution

- What is the output of the following code?

```
i, sum = 0, 0
while i <= 4:
    sum += i
    i += 1
print(sum)
```

- ☐ 0
- ☑ 10
- ☐ 4
- ☐ none of the above

```
i, sum = 0, 0
while i >= 4:
    sum += i
    i += 1
print(sum)
```

- ☑ 0
- ☐ 10
- ☐ 4
- ☐ none of the above

Practice

Problem: The summation of n

- Find the summation from 1 to the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Hint: $1 + 2 + 3 + 4 + \cdots + n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	3	4	6
1	6	10	21

Problem Solution

- Find the summation from 1 to the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Hint: $1 + 2 + 3 + 4 + \cdots + n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	3	4	6
1	6	10	21

```
n = int(input())
i, res = 1, 0
while i <= n:
    res += i
    i += 1
print(res)
```

Problem: The power of 2

- Find the powers of 2 from 0 to the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases

Hint: $2^0\ 2^1\ 2^2\ 2^3 \ldots 2^n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	2	4	6
1	1 2 4	1 2 4 8 16	1 2 4 8 16 32 64

© Prepared by: Mohamed Ayman

Problem Solution

- Find the powers of 2 from 0 to the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases

Hint: $2^0\ 2^1\ 2^2\ 2^3 \ldots 2^n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	2	4	6
1	1 2 4	1 2 4 8 16	1 2 4 8 16 32 64

```
n = int(input())
i = 0
while i <= n:
    print(2**i, end = ' ')
    i += 1
```

Problem: The factors of n

- Find the factors of the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	5	8	12
1	1 5	1 2 4 8	1 2 3 4 6 12

Problem Solution

- Find the factors of the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	5	8	12
1	1 5	1 2 4 8	1 2 3 4 6 12

```
n = int(input())
i = 1
while i <= n:
    if n % i == 0:
        print(i, end = ' ')
    i += 1
```

Problem: Calculate factorial of n

- Find the factorial of the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases:

Hint: $n! = 1 * 2 * 3 * 4 * \ldots * n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	3	5	6
1	6	120	720

Problem Solution

- Find the factorial of the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases:

Hint: $n! = 1 * 2 * 3 * 4 * \ldots * n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	3	5	6
1	6	120	720

```
n = int(input())
i, res = 1, 1
while i <= n:
    res *= i
    i += 1
print(res)
```

Lecture Agenda

- ✔ Section 1: Loop Definition
- ✔ Section 2: While Loop Statements
- ✔ Section 3: Else with While Loop
- **Section 4: For Loop Statements**
- Section 5: Else with For Loop
- Section 6: Nested Loops
- Section 7: Loop Control Statements

© Prepared by: Mohamed Ayman

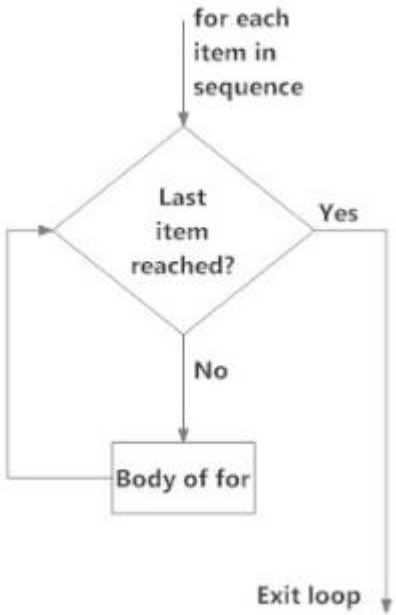
For Loop Statements

- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

```
for iterating_var in sequence:
    statement(s)
```

- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statement block is executed. Each item in the list is assigned to iterating_var, and the statement(s) blocks is executed until the entire sequence is exhausted.

- Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.



© Prepared by: Mohamed Ayman

For Loop Statements

Example:

```
for i in 'python':
    print(i)
print('the program completed')
```

Output:

```
p
y
t
h
o
n
the program completed
```

For Loop Statements

Iterations Trace:

1-
```
for i in 'python':
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    print(i)                    p
print('the program completed')
```

3-
```
for i in 'python':
    print(i)
print('the program completed')
```

4-
```
for i in 'python':
    print(i)                    y
print('the program completed')
```

5-
```
for i in 'python':
    print(i)
print('the program completed')
```

6-
```
for i in 'python':
    print(i)                    t
print('the program completed')
```

For Loop Statements

Iterations Trace:

1-
```
for i in 'python':
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    print(i)                    h
print('the program completed')
```

3-
```
for i in 'python':
    print(i)
print('the program completed')
```

4-
```
for i in 'python':
    print(i)                    o
print('the program completed')
```

5-
```
for i in 'python':
    print(i)
print('the program completed')
```

6-
```
for i in 'python':
    print(i)                    n
print('the program completed')
```

© Prepared by: Mohamed Ayman

For Loop Statements

Iterations Trace:

1-
```
for i in 'python':
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    print(i)
print('the program completed')
```

For Loop Statements

Example:

```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
print('the program completed')
```

Output:

```
banana
apple
orange
the program completed
```

© Prepared by: Mohamed Ayman

For Loop Statements

Iterations Trace:

1-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
print('the program completed')
```

2-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
print('the program completed')
```

3-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)                    banana
print('the program completed')
```

4-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
print('the program completed')
```

5-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)                        apple
print('the program completed')
```

© Prepared by: Mohamed Ayman

For Loop Statements

Iterations Trace:

1-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
print('the program completed')
```

2-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)                          orange
print('the program completed')
```

3-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
print('the program completed')
```

4-
```
fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
print('the program completed')
```

© Prepared by: Mohamed Ayman

The range() function

- Python has a built-in function, called range, which allows you to easily construct sequences of integers. The function range does not return a list, rather it returns a special range object. However, you can pass it to the list function to convert the range object into a list, if you need or want a list instead. The range function takes from 1 to 3 arguments. These arguments are converted into a start, stop, and step value as follows:
    - If a single argument is passed to range, the start value will be 0, the stop value will be the argument, and the step value will be 1.
    - If two arguments are passed to range, the start value will be the first argument, the stop value will be the second argument, and the step value will be 1.
    - If three arguments are passed to range, the start value will be the first argument, the stop value will be the second argument, and the step value will be the third argument.
- The range will then contain the sequence of integers which starts at the start value. Each successive value in the range will be the previous value in the range plus the step value. The range will continue up to (or down to, if the step is negative), but not including, the stop value.

© Prepared by: Mohamed Ayman

The range() function

- The built-in function range() is the right function to iterate over a sequence of numbers. It generates an iterator of arithmetic progressions.
- range() generates an iterator to progress integers starting with 0 up to n-1. To obtain a list object of the sequence, it is type casted to list(). Now this list can be iterated using the for statement. We can also define the start, stop and step size as range(start, stop, step). step size defaults to 1 if not provided.
- This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go. To force this function to output all the items, we can use the function list().

© Prepared by: Mohamed Ayman

The range() function

Example:

```
print(list(range(5)))
print(list(range(2, 7)))
print(list(range(3, 11, 2)))
print(list(range(8, 3, -1)))
```

Output:

```
[0, 1, 2, 3, 4]
[2, 3, 4, 5, 6]
[3, 5, 7, 9]
[8, 7, 6, 5, 4]
```

© Prepared by: Mohamed Ayman

The range() function

Example:

```
fruits = ['banana', 'apple', 'orange']
for i in range(len(fruits)):
    print(fruits[i])
print('the program completed')
```

Output:

```
banana
apple
orange
the program completed
```

The range() function

Example:

```
x = ['c', 'java', 'c++', 'python', 'scala', 'R', 'matlab']
for i in range(2, len(x)-2):
    print(x[i])
print('the program completed')
```

Output:

```
c++
python
scala
the program completed
```

Lecture Agenda

- ✔ Section 1: Loop Definition
- ✔ Section 2: While Loop Statements
- ✔ Section 3: Else with While Loop
- ✔ Section 4: For Loop Statements
- **Section 5: Else with For Loop**
- Section 6: Nested Loops
- Section 7: Loop Control Statements

Else with For Loop

- Python support having an else statement associated with a loop statement.
- If the else statement is used with a for loop, the else blocks is executed only if for loops terminates normally (and not by encountering break statement)
- A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.
- break statement can be used to stop a for loop. In such case, the else part is ignored. Hence, a for loop's else part runs if no break occurs.

```
for iterative_var in sequence:
    statement(s)
else:
    statement(s)
```

Else with For Loop

Example:

```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

Output:

```
48
67
73
we reach the end of the list
```

© Prepared by: Mohamed Ayman

Else with For Loop

Iterations Trace:

1-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

2-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

3-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])                    48
else:
    print('we reach the end of the list')
```

4-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

5-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])                    67
else:
    print('we reach the end of the list')
```

© Prepared by: Mohamed Ayman

Else with For Loop

Iterations Trace:

1-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

2-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])                    73
else:
    print('we reach the end of the list')
```

3-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

4-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

5-
```
x = [48, 67, 73]
for i in range(len(x)):
    print(x[i])
else:
    print('we reach the end of the list')
```

© Prepared by: Mohamed Ayman

Quiz

- What is the output of the following code?

```
x = [1, 2, [3, 4], 5]
for i in x:
    print(i)
```

☐
```
1
2
3
4
5
```

☐
```
1
2
[3, 4]
5
```

```
for i in [1, 0]:
    print(i+1)
```

☐
```
2
1
```

☐
```
[2, 1]
```

☐
```
2
0
```

☐
```
[2, 0]
```

Quiz Solution

- What is the output of the following code?

```
x = [1, 2, [3, 4], 5]
for i in x:
    print(i)
```

- [ ]
  ```
  1
  2
  3
  4
  5
  ```
- [x]
  ```
  1
  2
  [3, 4]
  5
  ```

```
for i in [1, 0]:
    print(i+1)
```

- [x]
  ```
  2
  1
  ```
- [ ]
  ```
  [2, 1]
  ```
- [ ]
  ```
  2
  0
  ```
- [ ]
  ```
  [2, 0]
  ```

Practice

Problem: The summation of n

- Find the summation from 1 to the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Hint: $1 + 2 + 3 + 4 + \cdots + n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	3	4	6
1	6	10	21

Problem Solution

- Find the summation from 1 to the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Hint: $1 + 2 + 3 + 4 + \cdots + n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	3	4	6
1	6	10	21

```
n = int(input())
res = 0
for i in range(1, n+1):
    res += i
print(res)
```

Problem: The power of 2

- Find the powers of 2 from 0 to the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases

Hint: $2^0\ 2^1\ 2^2\ 2^3 \ldots\ 2^n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	2	4	6
1	1 2 4	1 2 4 8 16	1 2 4 8 16 32 64

© Prepared by: Mohamed Ayman

Problem Solution

- Find the powers of 2 from 0 to the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases

Hint: $2^0\ 2^1\ 2^2\ 2^3 \ldots\ 2^n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	2	4	6
1	1 2 4	1 2 4 8 16	1 2 4 8 16 32 64

```
n = int(input())
for i in range(n+1):
    print(2**i, end = ' ')
```

Problem: The factors of n

- Find the factors of the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	5	8	12
1	1 5	1 2 4 8	1 2 3 4 6 12

Problem Solution

- Find the factors of the input number using python loops, such that the input number data type will be integer and positive.
- Test Cases

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	5	8	12
1	1 5	1 2 4 8	1 2 3 4 6 12

```
n = int(input())
for i in range(1, n+1):
    if n % i == 0:
        print(i, end = ' ')
```

Problem: Calculate factorial of n

- Find the factorial of the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases:

Hint: $n! = 1 * 2 * 3 * 4 * \ldots * n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	3	5	6
1	6	120	720

© Prepared by: Mohamed Ayman

Problem Solution

- Find the factorial of the input number using python loops, such that the input number data type will be integer and non-negative.
- Test Cases:

Hint: $n! = 1 * 2 * 3 * 4 * \ldots * n$

Test Case 1	Test Case 2	Test Case 3	Test Case 4
0	3	5	6
1	6	120	720

```
n = int(input())
res = 1
for i in range(1, n+1):
    res *= i
print(res)
```

Lecture Agenda

- ✔ Section 1: Loop Definition
- ✔ Section 2: While Loop Statements
- ✔ Section 3: Else with While Loop
- ✔ Section 4: For Loop Statements
- ✔ Section 5: Else with For Loop
- **Section 6: Nested Loops**
- Section 7: Loop Control Statements

Nested loops

- Python programming language allows the use of one loop inside another loop.

```
while expression:
    while expression:
        statement(s)
    statement(s)

for iterating_var in sequence:
    for iterating_var in sequence:
        statement(s)
    statement(s)
```

Nested loops

Example:

```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

Output:

```
1 2 3
2 4 6
3 6 9
```

© Prepared by: Mohamed Ayman

Nested loops

Iterations Trace:

i = 1 j = 1

1-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

2-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

3-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

4-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

Nested loops

Iterations Trace:

i = 1 j = 2

1-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

2-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

3-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

4-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

© Prepared by: Mohamed Ayman

Nested loops

Iterations Trace:

i = 1 j = 3

1-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

2-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

3-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

4-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

© Prepared by: Mohamed Ayman

Nested loops

Iterations Trace:

i = 2 j = 4

1-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

2-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

3-
```
i = 1
while i < 4:
    j = 1
    while j < 4:
        print(i*j, end = ' ')
        j += 1
    print()
    i += 1
```

and so on ...

Nested loops

Example:

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

Output:

```
1 2 3
2 4 6
3 6 9
```

© Prepared by: Mohamed Ayman

Nested loops

Iterations Trace:

i = 1 j = 1

1-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

2-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

3-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

4-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

© Prepared by: Mohamed Ayman

Nested loops

Iterations Trace:

i = 1 j = 2

1-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

2-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

3-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

4-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

Nested loops

Iterations Trace:

i = 1 j = 3

1-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

2-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

3-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

4-
```
for i in range(1, 4):
    for j in range(1, 4):
        print(i*j, end = ' ')
    print()
```

and so on ...

Quiz

- What is the output of the following code?

```
for i in ['c++', 'python', 'java']:
    for j in ['easy', 'hard']:
        print(i, j)
```

```
x = ['c++', 'python', 'java']
y = ['easy', 'hard']
i = 0
while i < len(x):
    j = 0
    while j < len(y):
        print(x[i], y[j])
        j += 1
    i += 1
```

- ☐ c++ easy
  c++ hard
- ☐ c++ easy
  c++ hard
  python easy
  python hard
  java easy
  java hard
- ☐ c++ easy
  c++ hard
- ☐ c++ easy
  c++ hard
  python easy
  python hard
  java easy
  java hard
- ☐ c++ easy
  python easy
  java easy
- ☐ c++ easy
  python easy
  java easy

Quiz Solution

- What is the output of the following code?

```
for i in ['c++', 'python', 'java']:
    for j in ['easy', 'hard']:
        print(i, j)
```

- ☐ c++ easy
  c++ hard
  python easy
  python hard
  java easy
  java hard
- ☑ c++ easy
  c++ hard
  python easy
  python hard
  java easy
  java hard

```
x = ['c++', 'python', 'java']
y = ['easy', 'hard']
i = 0
while i < len(x):
    j = 0
    while j < len(y):
        print(x[i], y[j])
        j += 1
    i += 1
```

- ☐ c++ easy
  c++ hard
  python easy
  python hard
  java easy
  java hard
- ☑ c++ easy
  c++ hard
  python easy
  python hard
  java easy
  java hard

© Prepared by: Mohamed Ayman

Practice

Problem: Identity Matrix

- Print the Identity Matrix $\mathbf{I}$ with size n*n using python loops, such that the input number data type will be integer and positive.

Test Case 1

```
2

1 0
0 1
```

Test Case 2

```
3

1 0 0
0 1 0
0 0 1
```

Test Case 3

```
4

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

Problem Solution

- Print the Identity Matrix $\mathbf{I}$ with size n*n using python loops, such that the input number data type will be integer and positive.

Test Case 1

```
2

1 0
0 1
```

Test Case 2

```
3

1 0 0
0 1 0
0 0 1
```

Test Case 3

```
4

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

```
n = int(input())
for i in range(n):
    for j in range(n):
        if i == j:
            print(1, end = ' ')
        else:
            print(0, end = ' ')
    print()
```

Problem Solution

- Print the Identity Matrix $\mathbf{I}$ with size n*n using python loops, such that the input number data type will be integer and positive.

Test Case 1

```
2

1 0
0 1
```

Test Case 2

```
3

1 0 0
0 1 0
0 0 1
```

Test Case 3

```
4

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

```
n = int(input())
i = 0
while i < n:
    j = 0
    while j < n:
        if i == j:
            print(1, end = ' ')
        else:
            print(0, end = ' ')
        j += 1
    print()
    i += 1
```

Lecture Agenda

- ✔ Section 1: Loop Definition
- ✔ Section 2: While Loop Statements
- ✔ Section 3: Else with While Loop
- ✔ Section 4: For Loop Statements
- ✔ Section 5: Else with For Loop
- ✔ Section 6: Nested Loops
- **Section 7: Loop Control Statements**

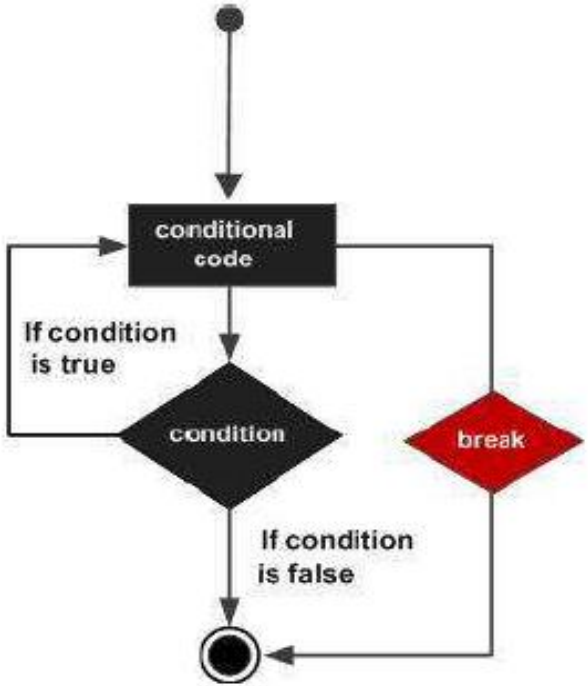
Loop Control Statements

- In Python, break and continue statements can alter the flow of a normal loop.
- Loops iterate over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.
- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

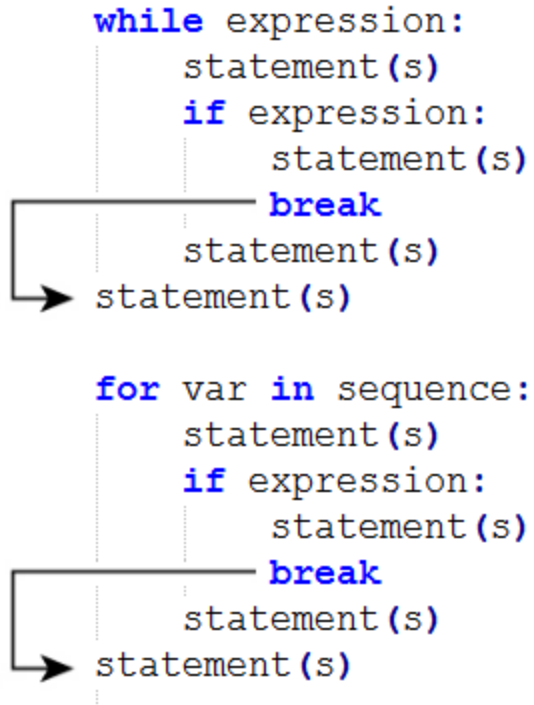
Loop Control Statements - break

- The break statement is used for premature termination of the current loop. After abandoning the loop, execution at the next statement is resumed.
- The most common use of break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.
- If you are using nested loops, the break statement stops execution of the innermost loop and starts executing the next line of the code after the block.



Loop Control Statements - break

- The working of break statement in for loop and while loop is shown below.



Loop Control Statements - break

Example:

```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

Output:

```
0
1
the program completed
```

© Prepared by: Mohamed Ayman

Loop Control Statements - break

Iterations Trace:

1-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

© Prepared by: Mohamed Ayman

Loop Control Statements - break

Iterations Trace:

1-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

2-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

3-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)                    0
    i += 1
print('the program completed')
```

4-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

Loop Control Statements - break

Iterations Trace:

1-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

2-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

3-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)                    1
    i += 1
print('the program completed')
```

4-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

© Prepared by: Mohamed Ayman

Loop Control Statements - break

Iterations Trace:

1-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

2-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

3-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

4-
```
i = 0
while i < 4:
    if i == 2:
        break
    print(i)
    i += 1
print('the program completed')
```

Loop Control Statements - break

Example:

```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

Output:

```
p
y
the program completed
```

Loop Control Statements - break

Iterations Trace:

1-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

3-
```
for i in 'python':
    if i == 't':
        break
    print(i)                    p
print('the program completed')
```

4-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

5-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

6-
```
for i in 'python':
    if i == 't':
        break
    print(i)                    y
print('the program completed')
```

Loop Control Statements - break

Iterations Trace:

1-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

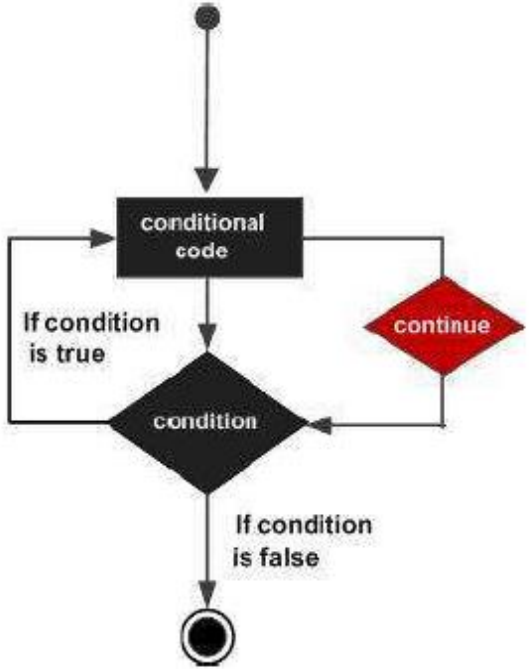
2-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

3-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

4-
```
for i in 'python':
    if i == 't':
        break
    print(i)
print('the program completed')
```

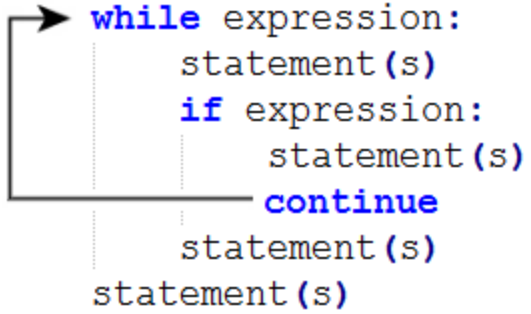
Loop Control Statements - continue

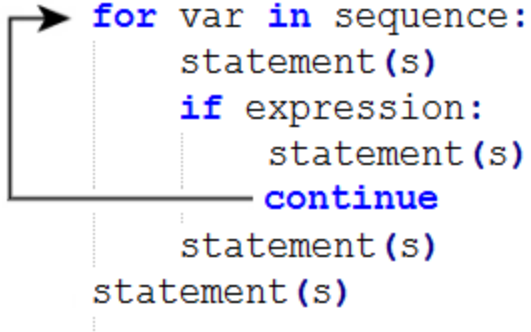
- The continue statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.
- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.



Loop Control Statements - continue

- The working of continue statement in for loop and while loop is shown below.





© Prepared by: Mohamed Ayman

Loop Control Statements - continue

Example:

```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

Output:

```
1
3
4
the program completed
```

Loop Control Statements - continue

Iterations Trace:

1-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

Loop Control Statements - continue

Iterations Trace:

1-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

2-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

3-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

4-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)                    1
print('the program completed')
```

Loop Control Statements - continue

Iterations Trace:

1-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

2-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

3-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

4-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

Loop Control Statements - continue

Iterations Trace:

1-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

2-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

3-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

4-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)                    3
print('the program completed')
```

Loop Control Statements - continue

Iterations Trace:

1-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

2-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

3-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

4-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)                    4
print('the program completed')
```

Loop Control Statements - continue

Iterations Trace:

1-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

2-
```
i = 0
while i < 4:
    i += 1
    if i == 2:
        continue
    print(i)
print('the program completed')
```

Loop Control Statements - continue

Example:

```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

Output:

```
p
y
h
o
n
the program completed
```

Loop Control Statements - continue

Iterations Trace:

1-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

3-
```
for i in 'python':
    if i == 't':
        continue
    print(i)                    p
print('the program completed')
```

4-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

5-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

6-
```
for i in 'python':
    if i == 't':
        continue
    print(i)                    y
print('the program completed')
```

© Prepared by: Mohamed Ayman

Loop Control Statements - continue

Iterations Trace:

1-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

3-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

4-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

5-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

6-
```
for i in 'python':
    if i == 't':
        continue
    print(i)                    h
print('the program completed')
```

© Prepared by: Mohamed Ayman

Loop Control Statements - continue

Iterations Trace:

1-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

3-
```
for i in 'python':
    if i == 't':
        continue
    print(i)                    o
print('the program completed')
```

4-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

5-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

6-
```
for i in 'python':
    if i == 't':
        continue
    print(i)                    n
print('the program completed')
```

© Prepared by: Mohamed Ayman

Loop Control Statements - continue

Iterations Trace:

1-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

2-
```
for i in 'python':
    if i == 't':
        continue
    print(i)
print('the program completed')
```

Loop Control Statements - pass

- In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored, however, nothing happens when pass is executed. It results into no operation.
- Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would complain. So, we use the pass statement to construct a body that does nothing.
- However, nothing happens when pass is executed. It results into no operation (NOP).

```
if expression:
    pass

while expression:
    pass

for var in sequence:
    pass
```

Loop Control Statements - pass

- It is used when a statement is required syntactically but you do not want any command or code to execute. The pass statement is a null operation nothing happens when it executes. The pass statement is also useful in places where your code will eventually go, but has not been written yet i.e. in stubs).

```
n = int(input())
if n < 0:
    pass

n = int(input())
for i in range(n):
    pass
```

Quiz

- Which of the following statement is true?

- ☐ The break statement terminates the loop containing it.
- ☐ The continue statement is used to skip the rest of the code inside the loop.
- ☐ The break and continue statements are almost always used with if, if...else and if...elif...else statements.
- ☐ All of the above.
- ☐ The Python interpreter ignores the pass statement like comments.
- ☐ The pass statement terminates the loop containing it.
- ☐ The pass is used as a placeholder for future implementation of functions, loops etc
- ☐ All of the above.

Quiz Solution

- Which of the following statement is true?

- [ ] The break statement terminates the loop containing it.
- [ ] The continue statement is used to skip the rest of the code inside the loop.
- [ ] The break and continue statements are almost always used with if, if...else and if...elif...else statements.
- [x] All of the above.
- [ ] The Python interpreter ignores the pass statement like comments.
- [ ] The pass statement terminates the loop containing it.
- [x] The pass is used as a placeholder for future implementation of functions, loops etc
- [ ] All of the above.

Quiz

- What is the output of the following code?

```
for i in 'PYTHON STRING':
    if i == ' ':
        break
    print(i, end='')
    if i == 'O':
        continue
```

- ☐ PYTHON
- ☐ PYTHONSTRING
- ☐ PYTHN
- ☐ PYTHO

```
for i in 'PYTHON STRING':
    if i == ' ':
        continue
    print(i, end='')
    if i == 'O':
        break
```

- ☐ PYTHON
- ☐ PYTHONSTRING
- ☐ PYTHN
- ☐ PYTHO

© Prepared by: Mohamed Ayman

Quiz Solution

- What is the output of the following code?

```
for i in 'PYTHON STRING':
    if i == ' ':
        break
    print(i, end='')
    if i == 'O':
        continue
```

- [x] PYTHON
- [ ] PYTHONSTRING
- [ ] PYTHN
- [ ] PYTHO

```
for i in 'PYTHON STRING':
    if i == ' ':
        continue
    print(i, end='')
    if i == 'O':
        break
```

- [ ] PYTHON
- [ ] PYTHONSTRING
- [ ] PYTHN
- [x] PYTHO

© Prepared by: Mohamed Ayman

Quiz

- What is the output of the following code?

```
x = [11, 33, 22, 55, 66, 77]
for i in x:
    if i % 2 == 0:
        print('the list contains even numbers')
        break
else:
    print('the list does not contain even numbers')
```

- ☐ the list does not contain even numbers
- ☐ the list contains even numbers
  the list contains even numbers
- ☐ the list contains even numbers
- ☐ the list contains even numbers
  the list does not contain even numbers
- ☐ none of the above

© Prepared by: Mohamed Ayman

Quiz Solution

- What is the output of the following code?

```
x = [11, 33, 22, 55, 66, 77]
for i in x:
    if i % 2 == 0:
        print('the list contains even numbers')
        break
else:
    print('the list does not contain even numbers')
```

- ☐ the list does not contain even numbers
- ☐ the list contains even numbers
the list contains even numbers
- ☑ the list contains even numbers
- ☐ the list contains even numbers
the list does not contain even numbers
- ☐ none of the above

© Prepared by: Mohamed Ayman

Quiz

- What is the output of the following code?

```
x = [11, 33, 22, 55, 66, 77]
i = 0
while i < len(x):
    if x[i] % 2 == 0:
        break
        print('the list contains even numbers')
    i += 1
else:
    print('the list does not contain even numbers')
```

- ☐ the list does not contain even numbers
- ☐ the list contains even numbers
  the list contains even numbers
- ☐ the list contains even numbers
- ☐ the list contains even numbers
  the list does not contain even numbers
- ☐ none of the above

© Prepared by: Mohamed Ayman

Quiz Solution

- What is the output of the following code?

```
x = [11, 33, 22, 55, 66, 77]
i = 0
while i < len(x):
    if x[i] % 2 == 0:
        break
        print('the list contains even numbers')
    i += 1
else:
    print('the list does not contain even numbers')
```

- ☐ the list does not contain even numbers
- ☐ the list contains even numbers
the list contains even numbers
- ☐ the list contains even numbers
- ☐ the list contains even numbers
the list does not contain even numbers
- ☑ none of the above

© Prepared by: Mohamed Ayman

Practice

Problem: Prime number

- Find if the input number is prime or not using python loops, such that the input number data type will be integer and positive.

Test Case 1	Test Case 2	Test Case 3	Test Case 4
2	4	5	9
YES	NO	YES	NO

Problem Solution

- Find if the input number is prime or not using python loops, such that the input number data type will be integer and positive.

Test Case 1	Test Case 2	Test Case 3	Test Case 4
2	4	5	9
YES	NO	YES	NO

```
n = int(input())
i = 2
while i < n:
    if n % i == 0:
        print('NO')
        break
    i += 1
else:
    print('YES')
```

© Prepared by: Mohamed Ayman

Problem Solution

- Find if the input number is prime or not using python loops, such that the input number data type will be integer and positive.

Test Case 1	Test Case 2	Test Case 3	Test Case 4
2	4	5	9
YES	NO	YES	NO

```
n = int(input())
for i in range(2, n):
    if n % i == 0:
        print('NO')
        break
else:
    print('YES')
```

Lecture Agenda

- ✔ Section 1: Loop Definition
- ✔ Section 2: While Loop Statements
- ✔ Section 3: Else with While Loop
- ✔ Section 4: For Loop Statements
- ✔ Section 5: Else with For Loop
- ✔ Section 6: Nested Loops
- ✔ Section 7: Loop Control Statements


DO
MORE.