Python Programming 1

(CP1S02005E)

Białystok University of Technology
Faculty of Electrical Engineering

Industry Digitization, semester II
Academic year 2024/2025

Lecture no. 02 (12.03.2025)

Jarosław Forenc, PhD

Topics

- For loop, range() function
- Break and continue statements
- Lists and for loop
- While loop

Python - for loop

- The for loop is used to iterate through elements of a collection or another iterable object
- For loop syntax:

```
for element in collection:
    # code executed for each element
```

  - element - a variable that takes the values of elements from the collection during each iteration
  - collection - a list, tuple, dictionary, set, or another iterable collection
- During each iteration, element takes the value of the next item in the collection, and the specified instructions are executed for that value

Python - range() function

- The range() function is used to generate a series of integers. It is commonly used in iterations, especially in for loops

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

  - start - integer from which the series begins, by default it has the value 0 (optional parameter)
  - stop - integer at which the series ends (not included in the series)
  - step - step by which the series increases, by default it has the value 1 (optional parameter)

- Returns a range object that represents a sequence of integers
- Does not generate a list immediately but creates an object that produces numbers on demand

Python - for loop and range() function

- generating integers in the range [0, 4]

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
for i in range(5):
    print(i, end = " ")
print()
```

```
0 1 2 3 4
```

Python - for loop and range() function

- generating integers in the range [2, 4]

```
for i in range(2,5):
    print(i, end = " ")
```

```
2 3 4
```

- generating integers in the range [1, 10] with a step of 2

```
for i in range(1,11,2):
    print(i, end = " ")
```

```
1 3 5 7 9
```

Python - for loop and range() function

- invalid start and stop values will result in an empty sequence

```
for i in range(9,0):
    print(i, end = " ")
```

```

```

- the step value can be negative

```
for i in range(9,0,-1):
    print(i, end = " ")
```

```
9 8 7 6 5 4 3 2 1
```

Python - for loop and range() function

- generating n+1 real numbers from the interval [start, stop]

```
start = 0
stop = 1
n = 10
step = (stop - start) / n
for i in range(n + 1):
    value = start + i * step
    print(value)
```

```
0.0
0.1
0.2
0.30000000000000004
0.4
0.5
0.6000000000000001
0.7000000000000001
0.8
0.9
1.0
```

- multiple instructions can be executed inside a for loop (they must have the same level of indentation)

Python - for loop and break statement

- the break statement is used inside a for loop to stop its execution, regardless of how many iterations have been completed
- Example: finding the first number divisible by both 3 and 7 in the range [1, 100]

```
for number in range(1, 101):
    if number % 3 == 0 and number % 7 == 0:
        print("The found number is:", number)
        break
```

```
The found number is: 21
```

Python - for loop and break statement

- the break statement is used inside a for loop to stop its execution, regardless of how many iterations have been completed
- Example: finding the first number divisible by both 3 and 7 in the range [1, 100] (result without break)

```
for number in range(1, 101):
    if number % 3 == 0 and number % 7 == 0:
        print("The found number is:", number)
```

```
The found number is: 21
The found number is: 42
The found number is: 63
The found number is: 84
```

Python - for loop and continue statement

- the continue statement is used to skip the current iteration of a for loop and proceed to the next one
- Example: displaying only odd numbers in the range [1, 10]

```
for number in range(1, 11):
    if number % 2 == 0:
        continue
    print(number, end = " ")
```

```
1 3 5 7 9
```

Python - for loop, most common errors

- missing colon (:)

```
for i in range(5)
    print(i)
```

```
PS C:\Users\jaros> python -u "d:\MyApp.py"
  File "d:\MyApp.py", line 1
    for i in range(5)
                      ^
SyntaxError: expected ':'
```

Python - for loop, most common errors

- no indentation for statements inside the for loop

```
for i in range(5):
print(i)
```

```
PS C:\Users\jaros> python -u "d:\MyApp.py"
  File "d:\MyApp.py", line 2
    print(i)
    ^
IndentationError: expected an indented block
                after 'for' statement on line 1
```

Python - for loop, most common errors

- unnecessary indentation

```
for i in range(5):
    print(i)
    print("End")
```

```
0
End
1
End
2
End
3
End
4
End
```

Python - for loop, most common errors

- attempting to iterate over a non-iterable object

```
value = 10
for i in value:
    print(i)
```

```
PS C:\Users\jaros> python -u "d:\MyApp.py"
Traceback (most recent call last):
  File "d:\MyApp.py", line 2, in <module>
    for i in value:
TypeError: 'int' object is not iterable
```

Python - nesting for loops

- Example: multiplication table

```
for i in range(1, 11):
    for j in range(1, 11):
        result = i * j
        print(f"{result:4}", end=" ")
    print()
```

```
   1    2    3    4    5    6    7    8    9   10
   2    4    6    8   10   12   14   16   18   20
   3    6    9   12   15   18   21   24   27   30
   4    8   12   16   20   24   28   32   36   40
   5   10   15   20   25   30   35   40   45   50
   6   12   18   24   30   36   42   48   54   60
   7   14   21   28   35   42   49   56   63   70
   8   16   24   32   40   48   56   64   72   80
   9   18   27   36   45   54   63   72   81   90
  10   20   30   40   50   60   70   80   90  100
```

Python - lists

- List - a list is a data structure that stores a collection of elements in a specific order
  - o create a list, its elements are placed inside square brackets ([]), separated by commas

```
passive = ["resistor", "coil", "capacitor"]
my_list = [1, 2, 3, "a", "b", "c", 2.5]
active = []
```

  - lists are dynamic - elements can be added or removed during program execution
  - lists can contain elements of different data types (numbers, strings, other lists, tuples, dictionaries)
  - list elements do not have to be related to each other

Python - lists

- to display a list, you can use the print() function

```
passive = ["resistor", "coil", "capacitor"]
print(passive)
```

```
['resistor', 'coil', 'capacitor']
```

```
my_list = [1, 2, 3, "a", "b", "c", 2.5]
active = []
print(my_list)
print(active)
```

```
[1, 2, 3, 'a', 'b', 'c', 2.5]
[]
```

Python - lists

- list elements are indexed, with the first element having index 0, the second 1, and so on
- to access an element, use the list name followed by square brackets ([]) containing the element's index

```
passive = ["resistor", "coil", "capacitor"]
print(passive[0])
print(passive[1])
print(f"The passive element is: {passive[0]}")
```

```
resistor
coil
The passive element is: resistor
```

Python - lists

- the last element of a list has an additional index of -1

```
passive = ["resistor", "coil", "capacitor"]
print(f"The last element is: {passive[-1]}")
```

```
The last element is: capacitor
```

- index -2 refers to the second-to-last element of the list
- index -3 refers to the third-to-last element of the list

```
passive = ["resistor", "coil", "capacitor"]
print(passive[-2])
```

```
coil
```

Python - lists

- values of list elements can be modified

```
passive = ["resistor", "coil", "capacitor"]
passive[1] = "inductor"
print(passive)
```

```
['resistor', 'inductor', 'capacitor']
```

- adding an element to the end of the list - use the append() method

```
passive = ["resistor", "coil", "capacitor"]
passive.append("diode")
print(passive)
```

```
['resistor', 'coil', 'capacitor', 'diode']
```

Python - lists

- inserting an element at any position - use the insert() method

```
passive = ["resistor", "coil", "capacitor"]
passive.insert(0,"diode")
print(passive)
```

```
['diode', 'resistor', 'coil', 'capacitor']
```

- removing a list element by index - use the del function

```
passive = ["resistor", "coil", "capacitor"]
del passive[2]
print(passive)
```

```
['resistor', 'coil']
```

Python - lists

- removing and returning the last element - use the pop() method

```
passive = ["resistor", "coil", "capacitor"]
print(passive)
element = passive.pop()
print(f"Removed element: {element}")
print(passive)
```

```
['resistor', 'coil', 'capacitor']
Removed element: capacitor
['resistor', 'coil']
```

Python - lists

- the pop() method can also take an index as an argument to remove a specific element

```
passive = ["resistor", "coil", "capacitor"]
print(passive)
element = passive.pop(1)
print(f"Removed element: {element}")
print(passive)
```

```
['resistor', 'coil', 'capacitor']
Removed element: coil
['resistor', 'capacitor']
```

Python - lists

- removing an element by value - use the remove() method

```
passive = ["resistor", "coil", "capacitor"]
print(passive)
passive.remove("resistor")
print(passive)
```

```
['resistor', 'coil', 'capacitor']
['coil', 'capacitor']
```

- the remove() method only deletes the first occurrence of the element

Python - lists

- removing all elements from a list - use the clear() method

```
passive = ["resistor", "coil", "capacitor"]
passive.clear()
print(passive)
```

```
[]
```

- determining the size of a list - use the len() function

```
passive = ["resistor", "coil", "capacitor"]
number = len(passive)
print(f"Number of elements in a list is: {number}")
```

```
Number of elements in a list is: 3
```

Python - for loop and lists

- displaying list elements - in the variable element, successive elements of the passive list are stored

```
passive = ["resistor", "coil", "capacitor"]
for element in passive:
    print(element)
```

```
resistor
coil
capacitor
```

- in this method, we do not need to know the number of elements in the list beforehand

Python - for loop and lists

- by using the enumerate() function, we can obtain the indexes of list elements

```
passive = ["resistor", "coil", "capacitor"]
for index, element in enumerate(passive):
    print(f"Index: {index}, Value: {element}")
```

```
Index: 0, Value: resistor
Index: 1, Value: coil
Index: 2, Value: capacitor
```

Python - for loop and lists

- creating a list of numbers using the range() function

```
numbers = list(range(1,10))
for nr in numbers:
    print(nr)
```

```
1
2
3
4
5
6
7
8
9
```

Python - while loop

- The while loop is used to execute a block of code as long as the condition is met (evaluates to True)
- Syntax of the while loop:

```
while condition:
    # code executed as long as the condition is true
```

  - Example:

```
number = 1
while number <= 5:
    print(number)
    number = number + 1
```

```
1
2
3
4
5
```

Python - while loop and break statement

- the break statement is used inside a while loop to terminate its execution
- Example: Finding the first number divisible by both 3 and 7 in the range [1, 100]

```
number = 1
while number <= 100:
    if number % 3 == 0 and number % 7 == 0:
        print("The found number is:", number)
        break
    number += 1
```

```
The found number is: 21
```

Python - while loop and continue statement

- the continue statement is used to skip the current iteration of the while loop and return to the beginning of the loop
- Example: displaying only odd numbers from the range [1, 10]

```
number = 0
while number < 10:
    number = number + 1
    if number % 2 == 0:
        continue
    print(number, end = " ")
```

```
1 3 5 7 9
```

End of lecture no. 2

Thank you for your attention!