

# Loops

Prof.Dr. Bahadır AKTUĞ  
803400815021 Machine Learning with Python

*\*Compiled from sources given in the references.*

# Loops

---

- ▶ The loops in Python can be formed directly over the datatypes(lists, tuples, dictionaries, strings) or with “while/for” structures.
- ▶ When the number of repetition is known beforehand, “for” is with a loop variable or direct looping over datatypes is preferred.
- ▶ When the loop is to be executed with a specific condition, then “while” loop structure is preferred.
- ▶ Both “for” and “while” can be used in a nested structure.
- ▶ There are also other control commands for both loop commands:
  - ▶ continue
  - ▶ pass
  - ▶ break
- ▶ If a loop is formed over a sequence and the elements of the sequence are also sequences, then multiple loop variables can be used.

# Loops

---

## Directly with sequences:

Lists, tuples, sets, dictionaries and strings can be used to form loops.

```
>>> colors= set(["Yellow", "Blue", "Green"])
```

```
>>> for color in colors:
```

```
...     print(color)
```

```
...
```

```
Yellow
```

```
Blue
```

```
Green
```

# Loops

---

## Directly with sequences:

Lists, tuples, sets, dictionaries and strings can be used to form loops.

```
>>> classname = "EEE105"  
>>> for letter in classname:  
...     print(letter)  
...  
E  
E  
E  
1  
0  
5
```

# Some useful functions for loops

---

## **range()**

- ▶ range() function produces a “range” object which consists of integers within a specified interval.
- ▶ This object can then be transformed into another datatype (list, tuple etc.).
- ▶ The general form of the command is given below:

`range([start], end[, increment])`

- ▶ The parameters of the "range" command have to be integers.
- ▶ Since version 3, the range command in Python returns an “iterator” object.

# Some useful functions for loops

---

## **range()** examples:

```
>>> for i in range(5):  
...     print(i)  
...  
0 1 2 3 4
```

```
>>> list(range(0,10,3))  
[0, 3, 6, 9]
```

```
>>> list(range(5,10))  
[5, 6, 7, 8, 9]
```

# Some useful functions for loops

---

## **enumerate()**

- ▶ "enumerate" command produces an ordered index list for the elements of a sequence.
  - ▶ "enumerate" command actually produces an "enumerate" object consisting of integers within the specified interval.
  - ▶ "enumerate" object can be converted to any sequence type (lists, tuples etc.). The general form of the command is:
- 
- ▶ `enumerate(sequence [, start=0])`
- 
- ▶ The parameters of "enumerate" command must be integers.

# Some useful functions for loops

---

## **enumerate()** examples:

```
>>> choices = ['döner','adana','iskender','mantı']  
>>> list(enumerate(choices))  
[(0, 'döner'), (1, 'adana'), (2, 'iskender'), (3, 'mantı')]
```

```
>>> for index, item in enumerate(choices, start = 1):  
...     print(index, item)  
...  
1 döner  
2 adana  
3 iskender  
4 mantı
```



# Some useful functions for loops

---

## zip()

- ▶ "zip" function takes multiple sequences as input and glues them pairwise in an ordered manner.
- ▶ "zip" command produces a zip object within the dimensions of the input sequences.
- ▶ This object can be converted into another sequence (list, tuple etc.).
- ▶ The general form of the command is:

`zip(a, b [, c, d, ...])`

# Some useful functions for loops

---

## **zip() examples**

```
>>> zip(range(5), range(1,20,2))  
[(0, 1), (1, 3), (2, 5), (3, 7), (4, 9)]
```

```
>>> colors = ['red', 'green', 'blue']  
>>> vals = [55, 89, 144, 233]  
>>> for col, val in zip(colors, vals):  
...     print(col, val)  
(red, 55)  
(green, 89)  
(blue, 144)
```

# Some useful functions for loops

---

## «\*» operator (unzip)

- ▶ «\*» operator functions like unzip to a sequential data type

```
>> data = ["class", "Python", "script", "example"]
```

```
>> print(data)
```

```
["class", "Python", "script", "example"]
```

```
>> print(*data, sep=';')
```

```
class;Python;script;example
```

# Loops

---

## **for:**

When a fixed number of loops is desired, the “for” command can be used as follows:

```
>>> for i in range(5):  
...     print(i)  
...  
0  
1  
2  
3  
4
```

# Loops

---

## **while:**

- ▶ When the termination of a loop depends on a condition, “while” structure is preferred.
- ▶ There must be a “boolean” expression which evaluates to either “true” or “false” after “while”

```
>>> n = 3
>>> i = 0
>>> while i < n:
...     print(i)
...     i += 1
...
0
1
2
```

# break

---

- ▶ “break” is necessary to terminate a loop (either while or for) at a specific point.
- ▶ When there are nested loops, the innermost loop is terminated.

```
>>> for letter in "EEEI05":
```

```
...     if letter == 'I':
```

```
...         break
```

```
...     print(letter)
```

```
...
```

```
E
```

```
E
```

```
E
```

## continue

---

- ▶ When it is needed to return to the loop command and continue the loop with the next value, “continue” command is used (both for while and for).
- ▶ When the “continue” is used, the loop continues with next element (if any) and skips over the rest of loop block.

```
>>> for letter in “EEEI05”:
```

```
...     if letter == ‘I’:
```

```
...         continue
```

```
...     print(letter)
```

```
...
```

```
E
```

```
E
```

```
E
```

```
0
```

```
5
```

# Commands within Loops

---

## **pass**

- ▶ When it is need to fill in a command block which does not do anything, “pass” command can be used as a placeholder.

```
>>> for letter in "EEEI05":
```

```
...     if letter == '0':
```

```
...         pass
```

```
...     else:
```

```
...         print(letter)
```

```
...
```

```
E
```

```
E
```

```
E
```

```
I
```

```
5
```



# Logical Expressions

Prof.Dr. Bahadır AKTUĞ  
803400815021 Machine Learning with Python

*\*Compiled from sources given in the references.*

# Logical Comparisons

---

- ▶ Python, provides “if/elif/else” blocks for logical comparison as almost all other programming languages do.
- ▶ A logical comparison takes a boolean expression (which evaluates either True or False) as input.
- ▶ Boolean expressions:
- ▶ The following are assumed “false” :
  - ▶ All numbers with a value of zero
  - ▶ Boolean “False”,
  - ▶ Empty string variables,
  - ▶ Empty lists/tuples/dictionaries
  - ▶ None value
- ▶ Everything else (values, variables) is assumed true

# if / elif / else

---

The general form of “if” is given below. “elif” command can be repeated as many as needed.

if condition-1:

    Command Block

elif condition-2:

    Command Block

elif condition-3:

    Command Block

else:

    Command Block

# Ternary if

---

A ternary operator is an operator which has three operands. The ternary "if" is expressed as follows:

```
max = a if a > b else b
```

This command can be expressed with usual if/else blocks as follows:

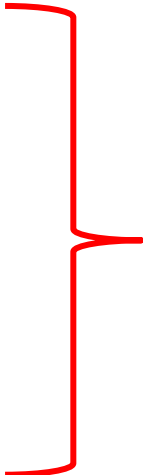
```
if a > b:  
    max=a  
else:  
    max=b
```

# Ternary if

---

Ternary if can even be extended to substitute several conditional statements:

```
if a > 5:  
    b = 100  
elif a == 5:  
    b = 10  
else:  
    b = 0
```



Original code block

The short ternary code:

```
b = 100 if a > 5 else 10 if a == 5 else 0
```

# match/case

---

- ▶ As opposed C, C#, Javascript, there is no switch/case command in Python.
- ▶ If/elif/else blocks were used to mimic switch case blocks.

```
day = 'Wednesday'

if day == 'Saturday':
    print('Weekend!')
elif day == 'Sunday':
    print('Weekend!')
else:
    print('A workday!')
```

# match/case

---

- ▶ However, starting from Python 3.10, match/case statement is available.

```
day = 'Wednesday'

match day:
    case 'Saturday':
        print('Weekend!')
    case 'Sunday':
        print('Weekend!')
    case _ :
        print('A workday!')
```

Underscore  
corresponds to the  
cases not matching  
others

# for/else

---

- ▶ There is a less known feature of “for” blocks in Python. “else” can be used within a “for” block.
- ▶ The function of “else” statement depends on whether there is a break in the “for” block.
- ▶ If there is no “break” statement, “else” is executed normally.

```
for day in ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']:
    print(day)
else:
    print('Working days listed!')
```

```
D:\Ders_Notlari\803400815021_MachineLearningWithPython>python forelse.py
Monday
Tuesday
Wednesday
Thursday
Friday
Working days listed!
```



# for/else

---

- ▶ When there is a “break” statement, then “else” block is not executed.

```
for day in ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']:
    print(day)
    break
else:
    print('Working days listed!')
```

```
D:\Ders_Notlari\803400815021_MachineLearningWithPython>python forelse.py
Monday
```

- ▶ This property is useful when you want to execute a block when any of the loop variable matches a given constant.

# for/else

---

- ▶ This property is useful when you want to check if any of the loop variable does not satisfy a specific condition.
- ▶ For instance, in the following example, to determine whether a number is not prime is easier than a prime number.

```
for n in range(2, 10):  
    for x in range(2, n):  
        if n % x == 0:  
            break  
    else:  
        print(n, 'is a prime number')
```

```
D:\Ders_Notlari\803400815021_MachineLearningWithPython>python forelse.py  
2 is a prime number  
3 is a prime number  
5 is a prime number  
7 is a prime number
```

# walrus operator (:=)

---

- ▶ Since Python 3.8, walrus operator is available. Walrus Operator allows you to assign a value to a variable within an expression.
- ▶ This can be useful when you want to simultaneously assign a value and use the value in logical expression.

```
numbers = [1, 2, 3, 4, 5]

while (n := len(numbers)) > 0:
    print(n)
    numbers.pop()
```

```
D:\Ders_Notlari\803400815021_MachineLearningWithPython>python walrus.py
5
4
3
2
1
```

---

## ► References

- 1 Wentworth, P., Elkner, J., Downey, A.B., Meyers, C. (2014). *How to Think Like a Computer Scientist: Learning with Python* (3rd edition).
- 2 Pilgrim, M. (2014). *Dive into Python 3* by. Free online version: [DiveIntoPython3.org](http://DiveIntoPython3.org) ISBN: 978-1430224150.
- 3 Summerfield, M. (2014) *Programming in Python 3 2nd ed (PIP3)* :- Addison Wesley ISBN: 0-321-68056-1.
- 4 Summerfield, M. (2014) *Programming in Python 3 2nd ed (PIP3)* :- Addison Wesley ISBN: 0-321-68056-1.
- 5 Jones E, Oliphant E, Peterson P, et al. *SciPy: Open Source Scientific Tools for Python*, 2001-, <http://www.scipy.org/>.
- 6 Millman, K.J., Aivazis, M. (2011). *Python for Scientists and Engineers, Computing in Science & Engineering*, 13, 9-12.
- 7 John D. Hunter (2007). *Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering*, 9, 90-95.
- 8 Travis E. Oliphant (2007). *Python for Scientific Computing, Computing in Science & Engineering*, 9, 10-20.
- 9 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). *Data Structures and Algorithms in Python*, Wiley.
- 10 <http://www.diveintopython.net/>
- 11 <https://docs.python.org/3/tutorial/>
- 12 <http://www.python-course.eu>
- 13 <https://developers.google.com/edu/python/>
- 14 <http://learnpythonthehardway.org/book/>