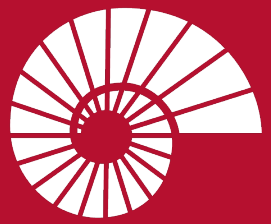


KOÇ
ÜNİVERSİTESİ

07 – Strings

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

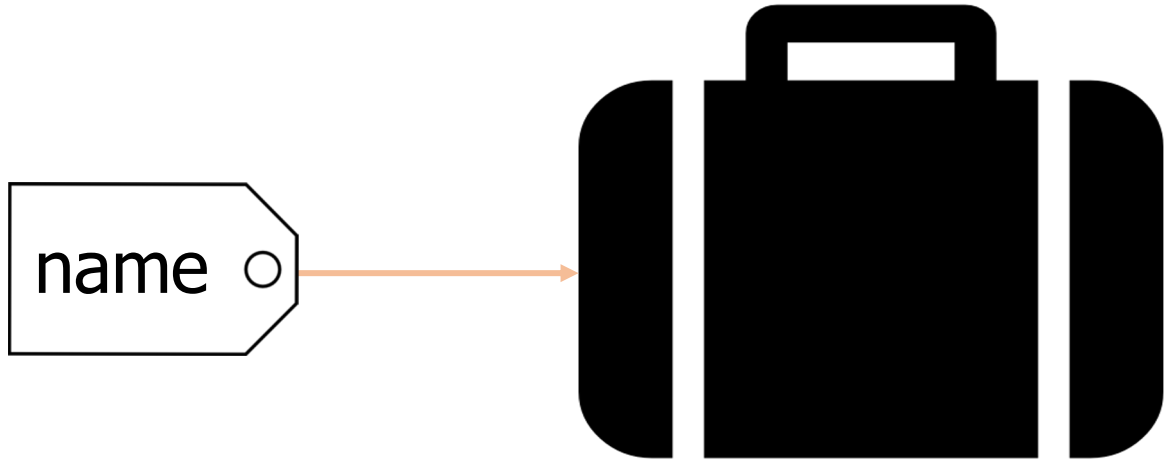
The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

Short Detour: Data Types

- We have seen
  - Integers
  - Floats
  - Booleans
- Sort of talked about strings here and there
- There are many other data types

Remember: The Suitcase Analogy

- When you store information in Python, it becomes a Python object
  - Objects come in different sizes and **types**
- You can think of a Python object as a suitcase stored in your computer's memory
- A variable is a luggage tag for your suitcase that gives it a name!



Remember: The Suitcase Analogy

- When you store information in Python, it becomes a Python object
  - Objects come in different sizes and **types**

Remember: The Suitcase Analogy

- When you store information in Python, it becomes a Python object
  - Objects come in different sizes and **types**

All Python Objects Have a `Type`

- Python automatically figures out the type based on the value
  - Variables are "**dynamically-typed**": you don't specify the type of the Python object they point to

- We have previously introduced the built-in Python function: `type`

- This returns the type of a Python object

Text Data – Characters

- A character is a minimal unit of text

- Letters of the alphabet, numbers, punctuation, white spaces, etc.
  - Digits: 0-9
  - Alphanumeric characters: a-z A-Z 0-9
  - White space characters: \n \t \r

- **In Python, there is no separate character data type!**

Text Data – Strings

- String literal is a sequence of characters enclosed in single (') or double quotes (")

```
my_name = 'Beren'
class_name = 'comp125'
sentence = "Comp125 is fun!"
```

Note: Use triple quotes for multi-line strings (also for multi-line comments)

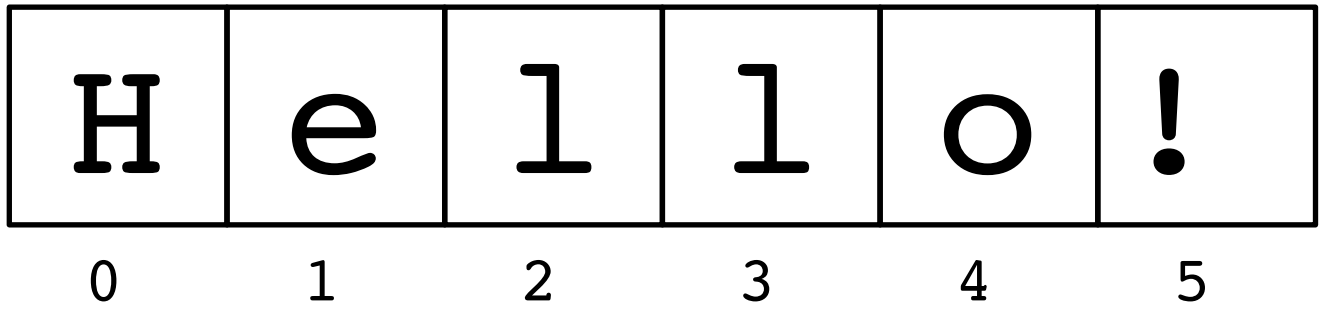
```
"""
This is a multi-line
string
"""
```

- The **length** of a string is the number of characters it contains
- Python function `len()` to evaluate the length of a string

```
len("Hello!") → 6
len('') → 0
len(class_name) → 7
```

Indexing

- Each character in the string is associated with an **`index`**
  - **index**: An integer representing the location of a character in a string
- Zero-based indexing
  - Indices start with 0 (not 1)
  - The **first character** of a string exists at **index 0**
  - The **last character** of a string exists at **index `len(s) - 1`**
  - Index **`len(s)`** is NOT a valid index



Indexing

- You can access a character in the sequence (string) by specifying its index in **square brackets [ ]**
- The character at index `i` of string `s` can be accessed with the expression `s[i]`

```
text = 'Hello!'
my_character = text[1]    # my_character = 'e'
print(text[4])            # displays o on the screen
```

- You MUST use valid index values

```
text = 'Hello!'
print(text[6])
```

```
  File "/Users/cigdem/Desktop/comp125/
line 2, in <module>
    print(text[6])

IndexError: string index out of range
```

Strings are Immutable

- After the strings are created, their characters cannot be modified through indexing
- Indexing with immutable sequences can only be used for **accessing**

```
my_string = 'COMP125'
print(my_string[5])                    # Works
my_character = my_string[6]            # Also works
my_string[6] = '0'                     # Gives an error
```

```
  File "/Users/cigdem/Desktop/comp125/py Files/untitled21
    my_string[6] = '0'

TypeError: 'str' object does not support item assignment
```

- To change a string, you must first build a new string and then re-assign the string variable

```
my_string = 'COMP125'
my_string = 'COMP105'
```

String Concatenation

- Strings can be combined with the **+** operator in a process called **concatenation**

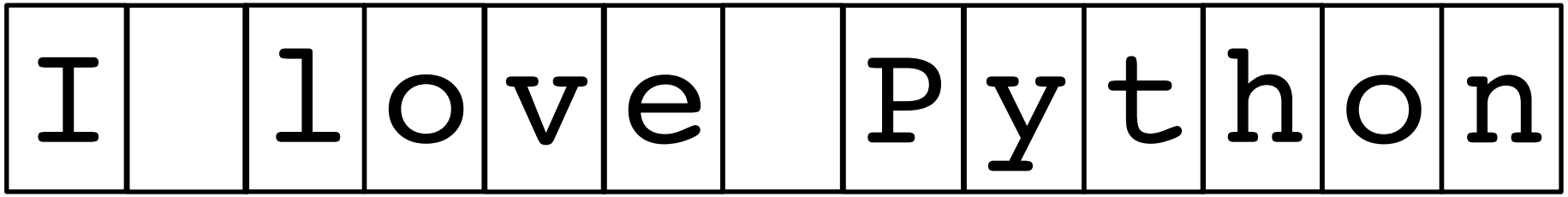
```
intro = 'Hello '
name = 'Nick'
greeting = intro + name   → 'Hello Nick'
greeting += '!'           → 'Hello Nick!'
```

- Concatenation does not change the existing string, but rather creates a new string and assigns the new string to the previously used variable

Slicing

- **Slice (or substring)** of a string is a consecutive block of characters that has been extracted from the original string
- Specify a range of indices in square brackets (remember the rules for the range function)

```
                    1         2
0123456789012345678901234
alphabet = 'abcdefghijklmnopqrstuvwxyz'
s1 = alphabet[3:6]        → def
s2 = alphabet[3:1]        → empty substring
s3 = alphabet[3:12]       → defghijkl
s4 = alphabet[3:12:2]     → dfhjl
```




```
string = 'I love Python'
print(string)
print(string[2:8])
print(string[2:])
print(string[2:len(string)])
print(string[2:-1])
print(string[2:20])
print(string[:-1])
print(string[4:-1])
print(string[-4:])
print(string[:])
print(string[0:10:2])
print(string[:10:2])
print(string[-2:3:-2])
print(string[9:-7])
```

```
I love Python
love P
love Python
love Python
love Pytho
love Python
I love Pytho
ve Pytho
thon
I love Python
Ilv y
Ilv y
otPe

```

nothing

Exercise

- Write a function that takes an input string and a character and returns how many times the character occurs within the input string

```
def how_many_times(my_str, my_char):
    occurrence = 0
    for i in range(len(my_str)):
        if my_str[i] == my_char:
            occurrence += 1
    return occurrence

def main():
    s = input('Enter your string: ')
    c = input('Enter your character: ')
    count = how_many_times(s, c)
    print(count)

main()
```

All Python Objects Have Attributes and Methods

- An object has attributes (properties, data) and methods (functionality) associated with them.
  - Ex: Object: Student; Attribute: Name; Attribute: Student ID; Method: Study for an exam
  - Ex: Object: Cell phone; Attribute: Color; Attribute: Brand; Functionality: Phone call
- **Dot notation**: provides access to attributes and methods of an object
- Remember modules

```
In [1]: import random

In [2]: type(random)
Out[2]: module

In [3]: type(random.random)
Out[3]: builtin_function_or_method

In [4]: random.random()
Out[4]: 0.7294606935235168
```

```
In [6]: import math

In [7]: type(math)
Out[7]: module

In [8]: type(math.pi)
Out[8]: float

In [9]: type(math.sqrt)
Out[9]: builtin_function_or_method
```

Basic String Methods

- `str.isupper()`
  - Returns True if all the characters in str are uppercase, False otherwise
- `str.islower()`
  - Returns True if all the characters in str are lowercase, False otherwise
- `str.isalpha()`
  - Returns True if all the characters in str are letters ('a'-'z', 'A'-'Z'), False otherwise
- `str.isdigit()`
  - Returns True if all the characters in str are digits ('0'-'9'), False otherwise
- `str.upper(), str.lower()`
  - Returns str with all letters converted to uppercase and lowercase, respectively
  - The original string remains unchanged

Exercise

- Suppose that a valid password should be of at least 10 characters and should contain at least one digit, one uppercase letter, and one lowercase letter. Write a function that takes a password as an input and returns true if it is a valid password and false otherwise.

```
def is_valid(password):
    has_digit = False
    has_lowercase = False
    has_uppercase = False
    for i in range(len(password)):
        if str.isdigit(password[i]):
            has_digit = True
        elif str.islower(password[i]):
            has_lowercase = True
        elif str.isupper(password[i]):
            has_uppercase = True

    return (has_digit and has_lowercase and
            has_uppercase and len(password) >= 10)

def main():
    p = input('Enter your password: ')
    valid_password = is_valid(p)
    if not valid_password:
        print('Invalid password')

main()
```

Exercise

- Suppose that you implement a computer game, in which you want to ask whether the user wants to continue once s/he plays and loses the game. The valid answers are yes and no (the letters may be lowercase or uppercase). You want to ask the user to enter her/his answer until s/he enters a valid answer.
- Write a function that takes the answer from the user and returns true if the user wants to continue and false otherwise.

```
def take_answer():
    answer = input('Do you want to continue? (yes or no): ')
    while str.upper(answer) != 'YES' and str.upper(answer) != 'NO':
        print('Invalid answer, enter again.')
        answer = input('Do you want to continue? (yes or no): ')

    if str.upper(answer) == 'YES':
        return True
    return False
```

Searching

- `in` : to test if one string is contained in another one
- `not in` : to test if one string is not contained in another one

```
text = 'Four score and seven years ago'
if 'seven' in text:
    print('The string "seven" was found.')
else:
    print('The string "seven" was not found.')
```

```
names = 'Bill Joanne Susan Chris Juan Katie'
if 'Pierre' not in names:
    print('Pierre was not found.')
else:
    print('Pierre was found.')
```

Basic String Methods (for reference)

Table 8-2 String Modification Methods

Method	Description
`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
★ `lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the beginning of the string.
`lstrip(char)`	The *`char`* argument is a string containing a character. Returns a copy of the string with all instances of *`char`* that appear at the beginning of the string removed.
★ `rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the end of the string.
`rstrip(char)`	The *`char`* argument is a string containing a character. The method returns a copy of the string with all instances of *`char`* that appear at the end of the string removed.
★ `strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(char)`	Returns a copy of the string with all instances of *`char`* that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

Basic String Method List (for reference)

Table 8-3 Search and replace methods

Method	Description
`endswith(substring)`	The *substring* argument is a string. The method returns true if the string ends with *substring*.
`find(substring)`	The *substring* argument is a string. The method returns the lowest index in the string where *substring* is found. If *substring* is not found, the method returns –1.
`replace(old, new)`	The *old* and *new* arguments are both strings. The method returns a copy of the string with all instances of *old* replaced by *new*.
`startswith(substring)`	The *substring* argument is a string. The method returns true if the string starts with *substring*.

You can also specify where to start and end the search!

string.startswith(*value, start, end*)
string.endswith(*value, start, end*)
string.find(*value, start, end*)

```
string = 'I love Python'
new_string = string.replace('Python','Matlab')
print(new_string)
```

```
I love Matlab
```

```
filename = 'mycode.py'
if filename.endswith('.py'):
    print('That is the name of a Python Source file.')
else:
    print('Your file is not a Python Source file.')
```

```
That is the name of a Python Source file.
```

```
string = 'Four thousand four hundred fifty four'
position = string.find('four')
print('four is found at index', position)

position2 = string.find('our')
print('our is found at index', position2)

new_string = string.lower()
position3 = new_string.find('four')
print('four is found at index', position3)
```

```
four is found at index 14
our is found at index 1
four is found at index 0
```

Splitting a string: `str.split()`

```
values = 'one two three four'
print(values.split())

['one', 'two', 'three', 'four’]
```

Default separator is any whitespace

```
values = 'one$two$three$four'
print(values.split('$'))

['one', 'two', 'three', 'four']
```

Exercise

- Write a code fragment that asks the user to enter three exam grades and displays their average

```
grade1 = input('First exam grade: ')
grade1 = float(grade1)

grade2 = input('Second exam grade: ')
grade2 = float(grade2)

grade3 = input('Third exam grade: ')
grade3 = float(grade3)

grade_avg = (grade1 + grade2 + grade3) / 3
print('Exam average is ', grade_avg)
```

Exercise

- Now, write another version of this code that takes the exam grades from the same line

```
grade_line = input('Enter exam grades: ')

grade_list = grade_line.split()

grade1 = float(grade_list[0])
grade2 = float(grade_list[1])
grade3 = float(grade_list[2])

grade_avg = (grade1 + grade2 + grade3) / 3
print('Exam average is ', grade_avg)
```

Default separator is any whitespace

This part becomes clear after learning lists

Formatting

Old School Formatting vs An Improved String Formatting Syntax

Option #1: %-formatting

```
In [32]: name="Comp. 125"

In [33]: "Hello %s" % name
Out[33]: 'Hello Comp. 125'
```

If we update “name”

```
In [34]: name="Comp. 125 Sec. 2"

In [35]: "Hello %s" % name
Out[35]: 'Hello Comp. 125 Sec. 2'
```

Option #2: str.format()

```
In [43]: name="Greta"

In [44]: lastname="Thunberg"

In [45]: "Hello, {0}. You are doing a great job {0} {1}.".format(name, lastname)
Out[45]: 'Hello, Greta. You are doing a great job Greta Thunberg.'
```

Formatting

f-string: A new formatting approach

```
In [46]: f"Hello, {name}. You are doing a great job {name} {lastname}."
Out[46]: 'Hello, Greta. You are doing a great job Greta Thunberg.'
```

```
In [47]: name="Atlas"

In [48]: lastname="Sarrafoğlu"

In [49]: f"Hello, {name}. You are doing a great job {name} {lastname}."
Out[49]: 'Hello, Atlas. You are doing a great job Atlas Sarrafoğlu.'
```

How do you want to align your text?

```
In [73]: f"{name:<10}"
Out[73]: 'Atlas     '

In [74]: f"{name:>10}"
Out[74]: '     Atlas'

In [75]: f"{name:^10}"
Out[75]: '  Atlas   '
```

Formatting

f-string: Further control

Type declaration not required in f-string !

```
In [58]: import math

In [59]: f"{2 * math.pi}"
Out[59]: '6.283185307179586'

In [60]: f"{2 * math.pi:10.5}"
Out[60]: '    6.2832'
```

Old style formatting for floating point numbers

```
In [63]: "%f" % (2 * math.pi)
Out[63]: '6.283185'

In [64]: "%10.5f" % (2 * math.pi)
Out[64]: '   6.28319'
```

Switch to different formats:

```
In [5]: f"{2 * math.pi:e}"
Out[5]: '6.283185e+00'

In [6]: f"{2 * math.pi:g}"
Out[6]: '6.28319'
```

Formatting: Further Reading

- % (string format) operator
  - https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting
- Built-in format() function
  - https://docs.python.org/3.8/library/functions.html?highlight=format%20function#format
- str.format() method
  - https://docs.python.org/3.8/library/stdtypes.html?highlight=format#str.format
  - Format specifier
    - https://docs.python.org/3.8/library/string.html#formatspec
- Formatted Strings (f-string)
  - https://docs.python.org/3.8/reference/lexical_analysis.html#f-strings