Python Programming 1

(CP1S02005E)

Białystok University of Technology

Faculty of Electrical Engineering

Industry Digitization, semester II

Academic year 2024/2025

Lecture no. 03 (19.03.2025)

Jarosław Forenc, PhD

Topics

- Strings
  - implementation, notation
  - element indexing
  - methods
  - string comparison
  - using the + and * operators

Python - strings

- A text string is a sequence of characters used to store textual information (data)
- A string can be enclosed in double quotes or single quotes

```
string1 = "Hello, world!"
string2 = 'Hello, world!'

string3 = """Hello, world!"""
string4 = '''Hello, world!'''
```

  - when enclosed with triple quotes, a string can span multiple lines of code

- Text strings are objects of the str class, more information:
  - https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python - strings

- you cannot use both double and single quotes at the same time to enclose a string

```
string1 = "Hello, world!'
```

```
File "d:\MyApp.py", line 1
    string1 = "Hello, world!'
              ^
SyntaxError: unterminated string literal (detected at line 1)
```

```
string1 = 'Hello, world!"
```

```
File "d:\MyApp.py", line 1
    string1 = 'Hello, world!"
              ^
SyntaxError: unterminated string literal (detected at line 1)
```

Python - strings

- other quotation marks (single/double quotes) can be used for quotations within a string

```
code1 = "Course 'C Programming' (CP1S01005)"
code2 = 'Course "Python Programming 1" (CP1S02005)'
```

- including a double quote (") inside a string enclosed by double quotes requires adding a backslash (\)

```
code1 = "Course \"C Programming\" (CP1S01005)"
```

- including a single quote (') inside a string enclosed by single quotes also requires adding a backslash (\)

```
code2 = 'Course \'Python Programming 1\' (CP1S02005)'
```

Python - strings

- example of an apostrophe in a string

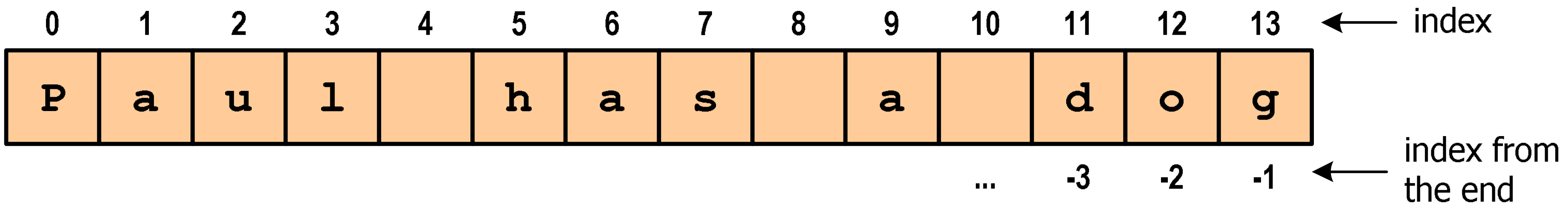
```
text1 = "Ampere's Law"
text2 = "It's a beautiful day"

print(text1)
print(text2)
```

```
Ampere's Law
It's a beautiful day
```

Python - strings (element indexing)

- a string is a sequential type, meaning that access to any element is possible by specifying its index

```
text = "Paul has a dog"
```



- indexing takes the form: string_name[index]

```
text = "Paul has a dog"

print(f"1st char from the beginning: {text[0]}")
print(f"2nd char form the begening:  {text[1]}")
print(f"2nd char from the end:       {text[-2]}")
```

```
P
a
o
```

Python - strings (element indexing)

- using a colon (:) to create element indexing

```
list[start_index : end_index : step]
```

- each index can be omitted, in which case the default values are assumed:

start_index:	0	(index of the first element)
end_index:	len(lista)	(index after the last element)
step:	1	

```
text = "Paul has a dog"
print(f"First three characters: {text[0:3]}")   # 0,1,2
```

```
First three characters: Pau
```

Python - strings (element indexing)

```
text = "Paul has a dog"
print(f"First three characters: {text[:3]}")
print(f"Text without the first three chars: {text[3:]}")
```

```
First three characters: Pau
Text without the first three chars:  l has a dog
```

```
text = "Paul has a dog"
print(f"Last character: {text[-1]}")
print(f"Last two characters: {text[-2:]}")
print(f"Without the last two characters: {text[0:-2]}")
```

```
Last character: g
Last two characters: og
Without the last two characters: Paul has a d
```

Python - strings (element indexing)

```python
text = "Paul has a dog"
print(f"Every 2nd element from the first: {text[::2]}")
print(f"Every 2nd element from the second: {text[1::2]}")
```

```
Every 2nd element from the first: Pu a  o
Every 2nd element from the second: alhsadg
```

```python
text = "Paul has a dog"
print(f"Reversed text: {text[::-1]}")
```

```
Reversed text: god a sah luaP
```

Python - strings (methods)

- the len() functions - returns the length of a text string (number of characters)

```
text = input("Enter text: ")
length = len(text)
print(f"Number of entered characters: {length}")
```

```
Enter text: Python Programming 1
Number of entered characters: 20
```

Python - strings (methods)

- the title() method - converts the first letter of each word to uppercase (does not modify the original text)

```
text = "paul has a dog"
print(text.title())
print(text)
```

```
Paul Has A Dog
paul has a dog
```

- permanent capitalization change

```
text = "paul has a dog"
text = text.title()
print(text)
```

```
Paul Has A Dog
```

Python - strings (methods)

- the upper() method - converts all lowercase letters to uppercase

```
text = "Paul Has a Dog"
print(text.upper())
```

```
PAUL HAS A DOG
```

- the lower() method - converts all uppercase letters to lowercase

```
text = "Paul Has a Dog"
print(text.lower())
```

```
paul has a dog
```

Python - strings (methods)

- the removeprefix() method - removes a prefix from a string (if it exists)

```
url = "https://we.pb.edu.pl"
url = url.removeprefix("https://")
print(url)
```

```
we.pb.edu.pl
```

- the removesuffix() method - removes a suffix from a string (if it exists)

```
fname = "grade_python.txt"
print(f"File name: {fname.removesuffix(".txt")}")
```

```
File name: grade_python
```

Python - strings (methods)

- the lstrip() method - removes whitespace from the left side of the string (beginning)
- the rstrip() method - removes whitespace from the right side of the string (end)
- the strip() method - removes whitespace from both sides of the string (beginning and end)
- whitespace characters: spaces (" "), tabs ("\t"), newline characters ("\n")

```python
text = "   John Smith  "

print(f"[{text}]")
print(f"[{text.lstrip()}]")
print(f"[{text.rstrip()}]")
print(f"[{text.strip()}]")
```

```
[   John Smith  ]
[John Smith  ]
[   John Smith]
[John Smith]
```

Python - strings (methods)

- the lstrip(), rstrip(), and strip() methods can take an argument specifying a set of characters to remove

```
text = "###John Smiths###"

print(f"[{text}]")
print(f"[{text.lstrip("#")}]")
print(f"[{text.rstrip("#")}]")
print(f"[{text.strip("#")}]")
```

```
[###John Smiths###]
[John Smiths###]
[###John Smiths]
[John Smiths]
```

Python - strings (methods)

- the startswith(prefix) method - returns True if the given string starts with the specified prefix

```
text = "Hello world"
if text.startswith("Hello"):
    print("The string starts with: 'Hello'")
else:
    print("The string does not start with: 'Hello'")
```

- the endswith(suffix) method - returns True if the given string ends with the specified suffix

```
text = "Hello world"
if text.endswith("world"):
    print("The string ends with: 'world'")
else:
    print("The string does not ends with: 'world'")
```

Python - strings (methods)

- the count(substring) method - returns the number of occurrences of a specified substring in the given string

```
text = "Paul has a dog"
cnt_a = text.count("a")
print(f"Number of occurrences of the letter 'a': {cnt_a}")
```

```
Number of occurrances of the letter 'a': 3
```

Python - strings (methods)

- the find(substring) method - searches for a specified substring in the given string
- returns the index of the first occurrence of the substring or -1 if the substring is not found

```
text = "Paul has a dog"
idx = text.find("dog")
if idx == -1:
    print("Substring not found")
else:
    print(f"Substring starts at index: {idx}")
```

```
Substring starts at index: 11
```

- the rfind(substring) method - returns the index of the last occurrence of the substring in the string or -1 if the substring is not found

Python - comparing strings

- comparison operators can be used with strings

Operator	Meaning	Operator	Meaning
==	equal to	!=	not equal to
>	greater than	>=	greater than or equal to
<	less than	<=	less than or equal to

- the result of a comparison is either True or False

```
pass = input("Enter a password: ")
if pass == "123456":
    print("This is the most popular password in the world")
else:
    print("The password is OK")
```

- two strings are considered equal if they consist of the same characters in the exact same positions

Python - comparing strings

- when comparing text, be mindful of letter case

```python
name = input("What was Einstein's first name? ")
if name.lower() == "albert":
    print("Correct answer!")
else:
    print("Incorrect answer!")
```

```
What was Einstein's first name? Albert
Correct answer!
```

```
What was Einstein's first name? albert
Correct answer!
```

```
What was Einstein's first name? Albie
Incorrect answer!
```

Python - strings (example)

- counting the number of digits in a string

```
text = (input("Enter text: "))
count = 0
for char in text:
    if ord(char) >= 48 and ord(char) <= 57:
        count = count + 1
print(f"The text contains {count} digits")
```

```
Enter text: asd58Dr4Hik2189
The text contains 7 digits
```

- the ord() function takes a single character (of type str) as an argument and returns its corresponding Unicode numerical value

Python - strings and operators: +, *

- the + operator is used to combine two strings into one

```
first_name = "John"
last_name = "Smith"
person = first_name + " " + last_name
print(person)
```

```
John Smith
```

- the * operator is used to repeat a string a specified number of times

```
text = "<->"
new_text = text * 10
print(new_text)
```

```
<-><-><-><-><-><-><-><-><-><->
```

Python - strings and lists

- the sort() method - sorts the elements in a list in alphabetical order; the sorting is permanent (modifies the original list)

```
passive = ["resistor", "inductor", "capacitor"]
passive.sort()
print(passive)
```

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

- elements can also be sorted in reverse alphabetical order

```
passive = ["resistor", "capacitor", "inductor"]
passive.sort(reverse = True)
print(passive)
```

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

Python - strings and lists

- the sorted() function - sorts the elements in a list in alphabetical order; does not modify the original list (returns a new sorted list)

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

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

- the sorted() function can also take the argument reverse=True for sorting in reverse alphabetical order

Python - strings and lists

- the split(separator, maxsplit) method splits a string into fragments (called tokens) based on a specified separator
- separator (optional) - a character or string used as the delimiter; by default, it splits at whitespace (spaces, tabs, newlines)
- maxsplit (optional) - the maximum number of splits to perform

```
text = input("Enter text: ")
tokens = text.split()
print(tokens)
```

```
Enter text: Paul has a dog
['Paul', 'has', 'a', 'dog']
```

End of lecture no. 3

Thank you for your attention!