Python Programming 1

(CP1S02005E)

Białystok University of Technology
Faculty of Electrical Engineering

Industry Digitization, semester II
Academic year 2024/2025

Lecture no. 05 (02.04.2025)

Jarosław Forenc, PhD

Topics

- Dictionary
  - implementation
  - creation methods
  - operations
- Set
  - implementation
  - creation methods
  - operations

Python - dictionary

- A dictionary is a collection of key-value pairs where each key is unique and associated with a specific value

```
descr = {"gender" : "M", "height" : 170, "eyes" : "gray"}
computer = {"processor" : "AMD", "disk" : "SSD"}
```

  - dictionary elements are enclosed in curly braces {}
  - a key is connected to its value using a colon :
  - key-value pairs are separated by commas
  - a key allows access to its corresponding value
  - keys can be numbers, strings, lists, or other dictionaries (any object that can be created in Python)
  - a dictionary can store any number of key-value pairs

Python - dictionary, creation methods

- using a pair of curly braces to create an empty dictionary

```
my_dict = {}
print(my_dict)
```

```
{}
```

- an empty dictionary is useful for storing user-inputted data or automatically generated key-value pairs
- you can also create an empty dictionary using the dict() constructor

```
my_dict = dict()
```

Python - dictionary, creation methods

- using curly braces, separating key-value pairs with commas, and connecting keys and values with a colon

```
computer = {"processor" : "Intel", "disk" : "HDD"}
letters = {"A" : 1, "B" : 2, "C" : 0}
numbers = {0 : 1, 1 : 1, 2 : 0}
print(computers)
print(letters)
print(numbers)
```

```
{'processor': 'Intel', 'disk': 'HDD'}
{'A': 1, 'B': 2, 'C': 0}
{0: 1, 1: 1, 2: 0}
```

Python - dictionary, creation methods

- the dictionary definition can be spread across multiple lines of code

```
computer = {
    "processor" : "Intel",
    "disk" : "HDD",
    "keyboard" : "A4Tech",
    "mouse" : "Logitech",
    }
```

```
{'processor': 'Intel', 'disk': 'HDD', 'keyboard':
'A4Tech', 'mouse': 'Logitech'}
```

- it is good practice to include a comma after the last key-value pair
- this ensures syntactic consistency and improves code readability

Python - dictionary, key uniqueness

- in a dictionary, each key must be unique
- if a value is assigned to an existing key, it will replace the previous value

```
age = {"John" : 21, "Alex" : 19, "John" : 18, "Kate" : 23}
print(age)
```

```
{'John': 18, 'Alex': 19, 'Kate': 23}
```

```
letters = {"A" : 1, "B" : 1, "B" : 2, "A" : 2}
print(letters)
```

```
{'A': 2, 'B': 2}
```

Python - dictionary, creation methods

- using dictionary comprehension
- dictionary comprehension refers to the syntax that allows creating a new dictionary by iterating over existing data and applying conditions
- general structure of dictionary comprehension

```
{key_expr: value_expr for item in iterable if condition}
```

- key_expr - an expression that defines the key for each key-value pair
- value_expr - an expression that defines the value for each key
- item - a variable used for iterating over elements in an iterable object
- iterable - an object that can be iterated over (e.g., list, tuple)
- condition - an optional condition that must be met for an element to be added to the dictionary

Python - dictionary, creation methods

- example: creating a dictionary from a list

```
numbers = [1, 2, 3, 4, 5]
squares = {num: num**2 for num in numbers}
print(squares)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

- example: creating a dictionary from a tuple

```
fruits = ('apple', 'banana', 'kiwi')
lengths = {fruit: len(fruit) for fruit in fruits}
print(lengths)
```

```
{'apple': 5, 'banana': 6, 'kiwi': 4}
```

Python - dictionary, creation methods

- example: creating a dictionary from a list with a condition

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = {num: num**2 for num in numbers if num % 2 == 0}
print(evens)
```

```
2: 4, 4: 16, 6: 36, 8: 64, 10: 100}
```

Python - dictionary, accessing values

- to retrieve a value associated with a key, use the dictionary name followed by the key name in square brackets

```
dictionary_name[key_name]
```

- example:

```
computer = {"processor" : "AMD", "disk" : "SSD"}
print(f"Processor: {computer["processor"]}")
print(f"Disk:      {computer["disk"]}")
```

```
Processor: AMD
Disk:      SSD
```

Python - dictionary, accessing values

- providing an incorrect key name will result in a runtime error

```
computer = {"processor" : "AMD", "disk" : "SSD"}
print(f"Processor: {computer["Processor"]}")
print(f"Disk:      {computer["disk"]}")
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 2, in <module>
    print(f"Processor: {computer["Processor"]}")
                        ~~~~~~~~^^^^^^^^^^^^^^
KeyError: 'Processor'
```

Python - dictionary, operations

- a dictionary is a dynamic structure, allowing key-value pairs to be added and removed at any time
- to add a new key-value pair, specify the dictionary name, the new key in square brackets, and the value to be assigned

```
computer = {"processor" : "AMD"}
print(computer)
computer["disk"] = "HDD"
computer["mouse"] = "A4Tech"
print(computer)
```

```
{'processor': 'AMD'}
{'processor': 'AMD', 'disk': 'HDD', 'mouse': 'A4Tech'}
```

- a dictionary maintains the order in which key-value pairs were added

Python - dictionary, operations

- if a key already exists, its value is updated

```
computer = {"processor" : "AMD", "disk" : "HDD"}
computer["disk"] = "SSD"
print(computer)
```

```
{'processor': 'AMD', 'disk': 'SSD'}
```

- you can also use the update() method to add new key-value pairs or update existing values

```
computer = {"processor" : "AMD"}
computer.update({"disk" : "SSD", "processor" : "Intel"})
print(computer)
```

```
{'processor': 'Intel', 'disk': 'SSD'}
```

Python - dictionary, operations

- the setdefault(key, default_value) method adds a new key with a specified default value if the key does not already exist in the dictionary

```
computer = {"processor" : "AMD"}
print(computer)
computer.setdefault("disk","SSD")
print(computer)
computer.setdefault("processor","Intel")
print(computer)
```

```
{'processor': 'AMD'}
{'processor': 'AMD', 'disk': 'SSD'}
{'processor': 'AMD', 'disk': 'SSD'}
```

Python - dictionary, operations

- the del function is used to remove a specific key from a dictionary

```
computer = {"processor" : "AMD", "disk" : "HDD"}
del computer["processor"]
print(computer)
```

```
{'disk': 'HDD'}
```

- the pop() method can be used to remove a key and return its corresponding value

```
computer = {"processor" : "AMD", "disk" : "HDD"}
result = computer.pop("disk")
print(computer)
print(result)
```

```
{'processor': 'AMD'}
HDD
```

Python - dictionary, operations

- in the pop() method, a default value can be provided as the second argument
- if the key does not exist, this value will be returned

```
computer = {"processor" : "AMD", "disk" : "HDD"}
result = computer.pop("disk","The key does not exist")
print(computer)
print(result)
result = computer.pop("mouse","The key does not exist")
print(computer)
print(result)
```

```
{'processor': 'AMD'}
HDD
{'processor': 'AMD'}
The key does not exist
```

Python - dictionary, operations

- the popitem() method is used to remove the last key-value pair from a dictionary
- this method removes and returns the last key-value pair as a tuple

```
computer = {"processor" : "AMD", "disk" : "HDD"}
key, value = computer.popitem()
print(computer)
print(key, value)
```

```
{'processor': 'AMD'}
disk HDD
```

- every key-value pair removal operation is irreversible

Python - dictionary, operations

- the get() method can be used to retrieve a value from a dictionary based on a given key

```
computer = {"processor" : "AMD", "disk" : "HDD"}
value = computer.get("disk")
print(value)
```

```
HDD
```

- if the key does not exist, the get() method returns a default value (None)

```
computer = {"processor" : "AMD", "disk" : "HDD"}
value = computer.get("mouse")
print(value)
```

```
None
```

Python - dictionary, operations

- checking values stored in a dictionary - comparison operator (==)

```
computer = {"processor" : "AMD", "disk" : "HDD"}
if computer["processor"] == "AMD":
    print("You have an AMD processor")
if computer["processor"] == "Intel":
    print("You have an Intel processor")
```

```
You have an AMD processor
```

Python - dictionary, operations

- iterating through all key-value pairs - use a for loop with the items() method

```
computer = {
    "processor" : "Intel",
    "disk" : "HDD",
    "mouse" : "Logitech",
    }
for key, value in computer.items():
    print(f"Key: {key}, value: {value}")
```

```
Key: processor, value: Intel
Key: disk, value: HDD
Key: mouse, value: Logitech
```

- such iteration returns a list of key-value pairs in the order they were inserted into the dictionary

Python - dictionary, operations

- the keys() method returns a view containing the dictionary's keys

```
computer = {
    "processor" : "Intel",
    "disk" : "HDD",
    "mouse" : "Logitech",
    }
for name in computer.keys():
    print(name)
```

```
processor
disk
mouse
```

- the same result can be obtained by omitting keys(), but the code becomes less explicit

```
for name in computer:
    print(name)
```

Python - dictionary, operations

- the values() method returns a view containing the dictionary's values

```
computer = {
    "processor" : "Intel",
    "disk" : "HDD",
    "mouse" : "Logitech",
    }
for name in computer.values():
    print(name)
```

```
Intel
HDD
Logitech
```

- the view can be converted into a list

```
my_list = list(computer.values())
print(my_list)
```

Python - dictionary, operations

- dictionary keys or values can be sorted during iteration

```
computer = {
    "processor" : "Intel",
    "disk" : "HDD",
    "mouse" : "Logitech",
    }
for name in sorted(computer.values()):
    print(name)
```

```
HDD
Intel
Logitech
```

- to ensure that displayed values do not repeat, a set should be used

```
for name in set(computer.values()):
    print(name)
```

Python - dictionary, operations

- nesting - a dictionary containing lists

```
programming = {
    "John" : ["C", "Python"],
    "Paul" : ["C", "C++", "Python"],
    "Mike" : ["Python"],
}

for name, languages in programming.items():
    if len(languages) == 1:
        print(f"{name} knows language:")
    else:
        print(f"{name} knows languages:"
    for language in languages:
        print(f"{language}", end = ", ")
    print("")
```

```
John knows languages:
C, Python,
Paul knows languages:
C, C++, Python,
Mike knows language:
Python,
```

Python - dictionary, operations

- nesting - a list containing dictionaries

```
PC1 = {"processor" : "AMD", "disk" : "SSD"}
PC2 = {"processor" : "Intel", "disk" : "HDD"}
PC3 = {"processor" : "Intel", "disk" : "SSD"}

computers = [PC1, PC2, PC3]

for PC in computers:
    print(PC)
```

```
{'processor': 'AMD', 'disk': 'SSD'}
{'processor': 'Intel', 'disk': 'HDD'}
{'processor': 'Intel', 'disk': 'SSD'}
```

- all dictionaries in the list should have the same structure to allow iteration over the list

Python - dictionary, operations

- operations on a dictionary and a list

```
cars = {
    "John" : "Opel",
    "Paul" : "BMW",
    "Mike" : "Audi",
    "Bart" : "BMW",
    }
colleagues = ["Paul", "Alex", "Mike", "George"]

for name in colleagues:
    if name not in cars.keys():
        print(f"{name} - no car")
    if name in cars.keys():
        print(f"{name} has a car: {cars[name]}")
```

```
Paul has a car: BMW
Alex - no car
Mike has a car: Audi
George - no car
```

Python - dictionary, data reading

- reading data from the keyboard directly into a dictionary

```
drivers = {}
reading = True

while reading:
    name = input("Enter the driver's name: ")
    brand = input("Enter the car brand: ")
    drivers[name] = brand

    ans = input("Continue? (yes/no): ")
    if ans == "no":
        reading = False

print("Data: ")
for name, brand in drivers.items():
    print(f"{name} has a {brand} car")
```

Python - dictionary, data reading

- reading data from the keyboard directly into a dictionary

```
drivers = {}
reading = True

while reading:
    name = input("Enter the
    brand = input("Enter th
    drivers[name] = brand

    ans = input("Continue?
    if ans == "no":
        reading = False

print("Data: ")
for name, brand in drivers.
    print(f"{name} has a {b
```

```
Enter the driver's name: Alex
Enter the car brand: BMW
Continue? (yes/no): yes
Enter the driver's name: Paul
Enter the car brand: Audi
Continue? (yes/no): yes
Enter the driver's name: Kate
Enter the car brand: KIA
Continue? (yes/no): no
Data:
Alex has a BMW car
Paul has a Audi car
Kate has a KIA car
```

Python - dictionary, nesting

- nesting a dictionary inside another dictionary

```
student = {
    "name" : "John",
    "surname" : "Smith",
    "grades" : {
        "Metrology" : 4.5,
        "Physics" : 3.0,
        "Mathematics" : 4.0
    }
}

print(f"Name: {student["name"]}")
print(f"Surname: {student["surname"]}")

print("Grades:")
for subject, grade in student["grades"].items():
    print(f"{subject}: {grade}")
```

```
Name: John
Surname: Smith
Grades:
Metrology: 4.5
Physics: 3.0
Mathematics: 4.0
```

Python - set

- Set - a collection of unique elements that are unordered
  - sets can be created using curly braces {} or the built-in set() function
  - creating an empty set:

```
empty1 = {}
print(empty1)
```

```
{}
```

```
empty2 = set()
print(empty2)
```

```
set()
```

Python - set, creation methods

- creating a set using curly braces

```
letters = {"A", "B", "C", "D", "E"}
print(letters)
```

```
{'D', 'A', 'B', 'E', 'C'}
```

- creating a set from a list of elements using the set() function

```
letters = set(["A", "B", "C", "C", "B"])
print(letters)
```

```
{'A', 'C', 'B'}
```

- duplicate elements will be automatically removed

Python - set, operations

- adding a single element to a set - add() method

```
numbers = {1, 2, 3, 4}
numbers.add(0)
print(numbers)
```

```
{0, 1, 2, 3, 4}
```

- adding multiple elements to a set - update() method

```
numbers = {1, 2, 3, 4}
numbers.update([0, 5, 6])
print(numbers)
```

```
{0, 1, 2, 3, 4, 5, 6}
```

Python - set, operations

- removing an element from a set - remove() method

```
numbers = {1, 2, 3, 4}
numbers.remove(1)
print(numbers)
```

```
{2, 3, 4}
```

- if the element does not exist in the set, remove() will raise an exception

```
numbers = {1, 2, 3, 4}
numbers.remove(0)
print(numbers)
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 2, in <module>
    numbers.remove(0)
KeyError: 0
```

Python - set, operations

- removing an element from a set - discard() method

```
numbers = {1, 2, 3, 4}
numbers.discard(1)
print(numbers)
```

```
{2, 3, 4}
```

- if the element does not exist in the set, discard() does not raise an error

```
numbers = {1, 2, 3, 4}
numbers.discard(0)
print(numbers)
```

```
{1, 2, 3, 4}
```

Python - set, mathematical operations

- union of two sets - union() method

```
numbers1 = {1, 2, 3, 4}
numbers2 = {3, 4, 5, 6}
sum12 = numbers1.union(numbers2)
print(sum12)
```

```
{1, 2, 3, 4, 5, 6}
```

Python - set, mathematical operations

- intersection of two sets - intersection() method

```
numbers1 = {1, 2, 3, 4}
numbers2 = {3, 4, 5, 6}
common12 = numbers1.intersection(numbers2)
print(common12)
```

```
{3, 4}
```

Python - set, mathematical operations

- difference of two sets - difference() method

```
numbers1 = {1, 2, 3, 4}
numbers2 = {3, 4, 5, 6}
diff12 = numbers1.difference(numbers2)
print(diff12)
```

```
{1, 2}
```

Python - set, mathematical operations

- symmetric difference of two sets - symmetric_difference() method

```
numbers1 = {1, 2, 3, 4}
numbers2 = {3, 4, 5, 6}
diff12 = numbers1. symmetric_difference(numbers2)
print(diff12)
```

```
{1, 2, 5, 6}
```

- this method returns a set containing elements that are present in only one of the two sets, but not in both at the same time

Python - set, in operator

- the in operator is used to check if an element is in a set

```
numbers = {1, 2, 3, 4, 5, 6}
num = int(input("Enter a number: "))

if num in numbers:
    print(f"Element {num} is in the set")
else:
    print(f"Element {num} is not in the set")
```

```
Enter a number: 6
Element 6 is in the set
```

```
Enter a number: 0
Element 0 is not in the set
```

Python - set, not in operator

- the not in operator is used to check if an element is not in a set

```
numbers = {1, 2, 3, 4, 5, 6}
num = int(input("Enter a number: "))

if num not in numbers:
    print(f"Element {num} is not in the set")
else:
    print(f"Element {num} is in the set")
```

```
Enter a number: 6
Element 6 is in the set
```

```
Enter a number: 0
Element 0 is not in the set
```

End of lecture no. 5

Thank you for your attention!