


# Programming in Python

Introduction to object-oriented programming
lecture 6

Department of Cybernetics and Artificial Intelligence
Technical University of Košice
Ing. Ján Magyar, PhD.

# Programming paradigms

- sequential programming
- procedural programming
- modular programming

- object-oriented programming

# Object-oriented programming

- the idea stems from the 70s
- rapid development after the arrival of Java
- the program is viewed not as a sequence of calls, but as a cooperation of independent blocks

# Why object-oriented programming?

- supporting modularity
- code reuse
- extending the language with your own data structures and types

# Data structure

- general description
- implemented as data types
- usually consists of a set of primitive values
- the goal is to increase the effectivity of working with data
- standard templates for representing data on computers
- different levels of abstraction
- we can implement it in multiple ways without altering the functionality

# Basic data structures

- arrays
- lists
- stack
- queue
- hash tables
- trees

# Stack

- dynamic set of values LIFO - last-in, first-out
- basic operations:
  - initialization - CREATE
    - creates an empty stack, sometimes part of PUSH
  - adding - PUSH
    - adds an element to the top of the stack
  - delete - POP
    - removes an element from the top of the stack
  - accessing the top of the stack - TOP
  - checking for values - IS_EMPTY

# Using stacks

- backtracking in algorithms looking for solutions in a trial-and-error way
  - depth-width search
- call stack

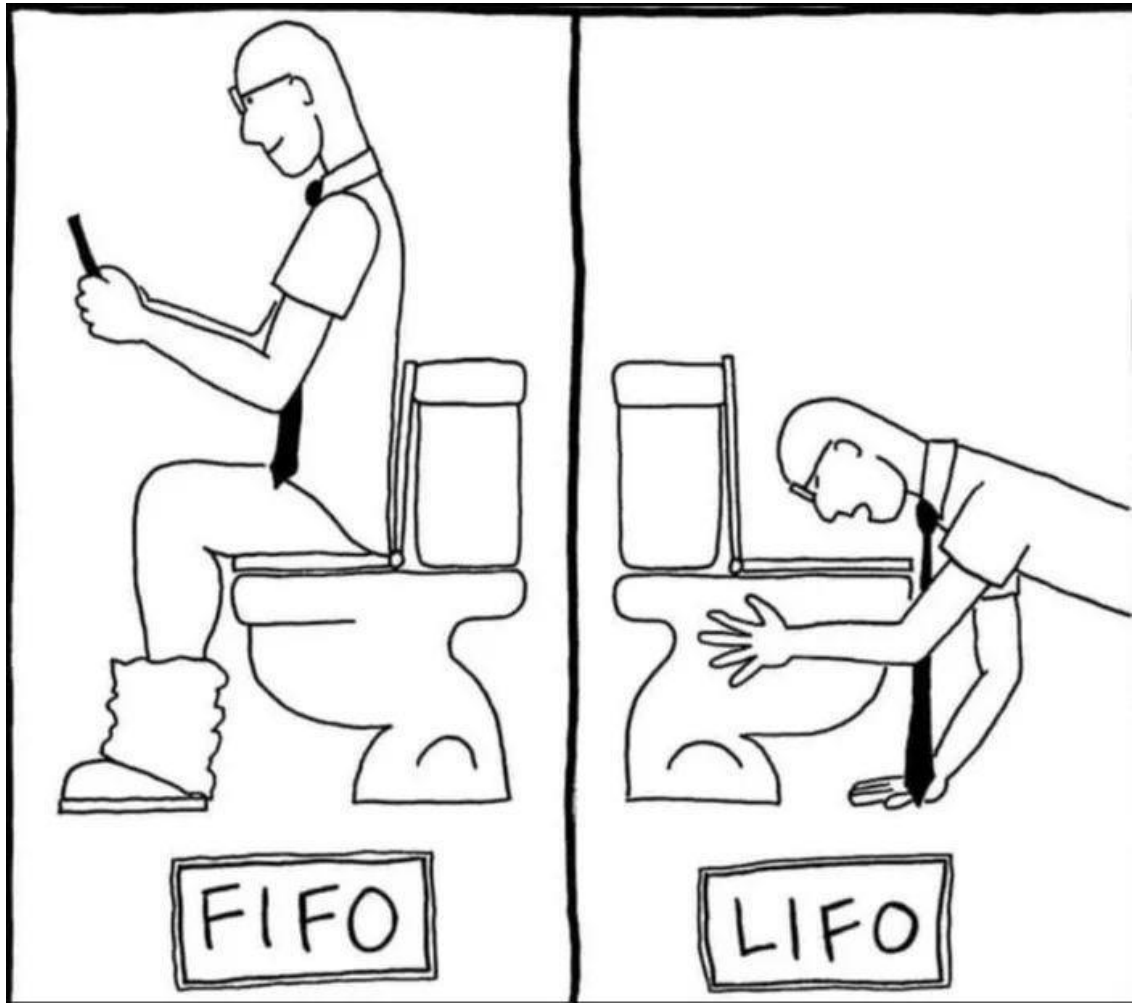
# Queue

- dynamic set of values FIFO - first-in, first-out
- basic operations:
  - initialization - CREATE
    - creates an empty queue, sometimes part of ENQUEUE
  - adding - ENQUEUE
    - adds an element to the end of the queue
  - delete - DEQUEUE
    - removes an element from the beginning of the queue
  - accessing the first element - HEAD
  - accessing the rest of the elements - TAIL
  - checking for values - IS_EMPTY

# Using queues

- processing requests
- communication between two processes
- sending messages
- temporary data storage for later processing

# LIFO/FIFO



# Structure of object-oriented solutions

- we split the code into modules
- a module is a group of intradependent functions and values
- the most common example of modules are libraries
- a solution is modular if parts of the code are split into multiple files

# Modules in Python

- if we want to work with modules, we must import them

```
import modul_name
```

- we can assign aliases to modules

```
import modul_name as name
```

- targeted import

```
from modul_name import this, that [as name]
```

# Solving name conflicts

- a name conflict arises when in the context of the code that is executed there are two attributes (usually functions) with the same name
- solution - dot notation

```
import numpy as np
import random

np.random.randint(5, 10)
random.randint(5, 10)
```

# Basic OOP constructs

- class and object
- a class is a template for creating objects
- an object is an instance of a class – concrete example

# Basic OOP constructs - class

- a **class** is a special type of modules
- used for data abstraction, it is an **abstract data type**
- connection to objects
  - it is a template for creating an object
  - a class is a collection of objects with the same characteristics
- from the point of view of the interpreter a class is a definition of a new data type
- in Python each class is also an object

# Basic OOP constructs - object

- an **object** is a class instance - concrete example/data/variable
- an object consists of **data** and **functions** (**methods**) - in Python both are considered attributes
- data is stored in fields - exist only inside the object

# Working with objects in Python

```
x = 5
y = 6
x += 1
y -= 2
```

```
my_lst = list()
my_lst.append(6)
my_lst.append(4)
```

# Defining a class in Python

```
class Product:
    def __init__(self):
        self.price = 1000

    def set_price(self, new):
        self.price = new

    def sell(self):
        print("Was sold for {}".format(self.price))
```

# Creating an object in Python

```
t = Product()
t.sell()


l = list()
l.append(5)
```

# Control flow in OOP

- with sequential programming, we view a program as a sequence of operations
- in OOP the program consists of objects sending messages

```
lst = list()
lst.append(5)
```

# Conclusion

- programming paradigms
- object-oriented programming
- data structures
- stack and queue
- modules
- class and object