



## Module 02 – Working with environment

# Agenda

- ✦ Introduction
- ✦ sys module
- ✦ Command-line arguments
- ✦ Standard data streams
- ✦ Redirections
- ✦ Exiting the program os module
- ✦ Environment variables
- ✦ Working with directories
- ✦ Process Information

# Introduction

- Working with the environment in Python refers to managing and manipulating the runtime environment in which a Python program executes.
- The environment includes various system-level variables, configurations, and resources that affect the behavior of the program.
- Python provides a module called `os` (Operating System) that allows developers to interact with the environment. It offers functions and methods to access and modify environment variables, work with files and directories, handle processes, and perform other system-related operations.

# sys module

- This module provides a number of functions and variables that can be used to manipulate different parts of the Python runtime environment.
- This module provides:
  - access to some variables used or maintained by the interpreter,
  - access to functions that interact with the interpreter

# sys module — version and platform

- `sys.version`
  - The output: 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)]

- `sys.platform`

<u>System</u>	<u>platform value</u>
Linux (2.x and 3.x)	linux2
Windows	win32
Windows/Cygwin	cygwin
etc	

# Command-line arguments

- Arguments passed to a script called Command Line Arguments
- Python script can access those command line arguments through `sys.argv` list.
- The first item of this list is a path of the script itself

```
import sys

for i, arg in enumerate(sys.argv):
    if i == 0:
        print("Script name: {}".format(sys.argv[0]))
    else:
        print("{} argument is: {}".format(i, sys.argv[i]))
```

# Standard data streams

- Almost every programmer familiar with standard streams:
  - standard input - as default, connected to the keyboard
  - standard output - as default, connected to the terminal (or working window)
  - standard error - as default, connected to the terminal (or working window)
- These data streams can be accessed from Python via the objects — of the `sys` module: `sys.stdin`, `sys.stdout` and `sys.stderr`.

Standard data streams

Demo





# Standard data streams — cont'd

```
import sys
```

```
sys.stdout.write("some string")
```

```
s="some string"
```

```
sys.stdout.write(s)
```

```
line = sys.stdin.readline()[:-1] #removes the \n from  
the end of the line sys.stdout.write(line)
```

# Redirections

- The standard output, error and input can be redirected e.g. into a file, so that we can process this file later with another program.
- We can redirect both stderr and stdout into the same file or into separate files

Console Methods

Demo



# Redirections — cont'd

```
import sys

fd = open("in", "r")
sys.stdin = fd

x = input()    #reads from
               the file
print(x)
```

```
import sys

save_stdout = sys.stdout
fd = open("test.txt", "w")
sys.stdout = fd
print("This line goes to test.txt")

#... Do things...

sys.stdout = save_stdout
fd.close() # return to default:
```

# os module

- The OS module provides a portable way of using operating system dependent functionality.
- The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on — be that Windows, Mac or Linux.

Lab 01

Lab



# Environment variables

- **os.environ** - A mapping object representing the string environment.
- **os.getenv(varname, value=None)** - Return the value of the environment variable varname if it exists, or value otherwise.  
For example:  

```
os.environ['HOME']  
os.getenv("HOME")
```
- Both returns a PATH environment variable value.
- PATH value specifies the directories in which executable programs 'are " located on the machine that can be started without knowing and typing the whole path to the file on the command line

# Environment variables — cont'd

- **os.putenv(varname, value)** - Set the environment variable named varname with a value.
  - Such a changes to the environment affects the sub-processes, created after the change



# os.path — Common pathname manipulations

- **os.path.dirname(path)** - return the directory name of pathname path
- **os.path.exists(path)** - return True if path refers to an existing path
- **os.path.isfile(path)** - return True if path is an existing regular file.
- **os.path.isdir(path)** - return True if path is an existing directory.
- **os.path.islink(path)** - return True if path refers to a directory entry that is a symbolic link
- **os.path.join(path, \*paths)** - join one or more path components intelligently
- **etc**

# Working with directories

- **os.getcwd()**— returns current working directory
- **os.chdir(path)**— change current directory
- Create a directory named path **os.mkdir(path)**
- Recursive directory creation function. **os.makedirs(path)**
- Return a list of the entries in the directory given by path. **os.listdir(path)**
- Remove (delete) the file path. **os.remove(path)**
- Remove (delete) the directory path. **os.rmdir(path)**

Lab 02

Lab



# Questions

