

# Lecture VI - Using Modules and Packages

Programming with Python

Dr. Tobias Vlček

# Quick Recap of the last Lecture

## Exceptions and Error Handling

- Exceptions are discovered errors during program execution
- Common built-in exceptions: `ValueError`, `TypeError`, etc.

...

```
x = int("Hello, World!")
```

...

>ValueError: invalid literal for int() with base 10: 'Hello, World!'

## Try-Except Blocks

- try-except blocks are used to handle exceptions
- try block contains code that might raise an exception
- except block contains code executed if an exception occurs

...

```
try:
    # Code that might raise an exception
    # ...
except ExceptionType as e:
    # Code to handle the exception
    # ...
except:
    # Code to handle any other exceptions
    # ...
```

## Raising Exceptions

- We can raise exceptions using the `raise` statement
- Allows for more controlled error handling
- Can include custom error messages

...

```
raise ValueError("This is a custom error message")
```

...

### Note

The type of raised exception **has to exist** or you have to create a custom error type before.

## Assertions

- Assertions check if a condition is true
- If the condition is false, an `AssertionError` is raised
- Useful for checking calculations or variable types

...

```
x = -1
assert x > 0, "x must be positive"
```

...

Question: Will this raise an `AssertionError`?

## Debugging

- Debugging is the process of finding and fixing errors in code
- Using `print` and `assert` statements
- Using logging
- Using built-in debugging tools in IDEs

...

### Tip

That's why IDEs are so helpful in coding.

# Modules

## Why Modules?

- Modular programming breaks large tasks into smaller subtasks
- Modules are like **building blocks** for larger applications
- Individual modules can be **combined** to create a complete program
- This approach enhances code organization and reusability

## Creating Modules

- Modules are simply .py files containing Python code
- They can define functions, classes, and variables
- They can be imported into other Python scripts

```
# The script new_module.py is in the same directory as this script
import new_module # Here we import the module
new_module.my_function() # Here we call the function from the module
```

Hello from my\_function!

## Importing functions from modules

- We can also import specific functions from a module
- This is useful if we only need a **few functions** from a module
- Analogously, we can import **classes or variables** from a module

...

```
# Multiple imports from a module are possible as well!
from new_module import another_function, yet_another_function
another_function()
yet_another_function()
```

Hello from another\_function!

Hello from yet\_another\_function!

...

### Tip

This is a good way to avoid importing too much from a module. In addition, we don't need to use the module name before the function name when we use the functions from the module.

## Built-in Modules

Python comes with many built-in modules. Common ones include:

| Module   | Description                      |
|----------|----------------------------------|
| math     | Different mathematical functions |
| random   | Random number generation         |
| datetime | Date and time manipulation       |
| os       | Operating system interaction     |
| csv      | Reading and writing CSV files    |
| re       | Regular expression operations    |

## Importing from the Standard Library

Task: Use Python's `math` module to calculate the area of a circle.

```
# Import the `math` module.
# Define a function named `calculate_area` that takes the radius `r` as an argument.
# Inside the function, use the `math.pi` constant to get the value of  $\pi$ .
# Calculate the area in the function and return it.
```

```
# Your code here
```

```
assert calculate_area(5) == 78.53981633974483
```

...

### Tip

Note, how assertions can be used to check if a function works correctly.

# Standard Libraries

## Random Numbers

The `random` module provides functions for random numbers

- `random.random()`: random float between 0 and 1
- `random.uniform(a, b)`: random float between a and b
- `random.randint(a, b)`: random integer between a and b
- `random.choice(list)`: random element from a list
- `random.shuffle(list)`: shuffle a list

### Tip

There are many more functions in the `random` module. Use the `help()` function to get more information about a module or function.

## Random Numbers in Action

Task: Time for a task! Import the `random` module and create a small number guessing game with the following requirements:

```
# Generate a random integer between 1 and 10 using randint().
# Ask the user to guess the number with input().
# Print whether the guess was correct.
# Give a hint if the guess was too high or too low.
# Repeat the game until the user guesses the number.

# Your code here
```

...

### Tip

Remember, that the `input` function always returns a string!

## OS Module

- The `os` module provides functions to interact with the OS
- `os.listdir(path)`: list all files and directories in a directory
- `os.path.isfile(path)`: check if a path is a file
- `os.path.isdir(path)`: check if a path is a directory

- `os.mkdir(path)`: create a directory

...

#### Tip

These can be quite useful for file handling. The `os` module contains many more functions, e.g. for changing the current working directory, for renaming and moving files, etc.

## CSV Module

- Comma-Separated Values files are used to store tabular data
- Write: `csv.writer(file)`
- Read: `csv.reader(file)`

...

```
import csv # Import the csv module

with open('secret_message.csv', 'w') as file: # Open the file in write mode
    writer = csv.writer(file) # Create a writer object
    writer.writerow(['Entry', 'Message']) # Write the header
    writer.writerow(['1', 'Do not open the file']) # Write the first row
    writer.writerow(['2', 'This is a secret message']) # Write the second row
```

...

Task: Copy the code and run it. Do you have a new file?

# Regular Expressions

## What are Regular Expressions?

- Regular expressions are a way to search for patterns in text
- They are a useful tool for string manipulation
- We can use the `re` module to work with regular expressions

```
...
import re
pattern = r'World' # This is the pattern we are searching for
string = 'Hello, World!' # This is the string we are searching in
print(re.search(pattern, string)) # This will search for the pattern in the string
```

```
<re.Match object; span=(7, 12), match='World'>
```

```
...
```

### Note

So far, we could also have achieved this with the `find` method of a string.

## Why Regular Expressions?

```
import re
pattern = 'World' # This is the pattern we are searching for
string = 'Hello, World!' # This is the string we are searching in
print(string.find(pattern)) # No regular expressions here!
```

```
7
```

```
...
```

- But regular expressions are more **powerful and flexible**
- They have special characters that allow for complex patterns
- They are widely used in text processing and web scraping

## Using Regular Expressions

- `re.search(pat, str)`: search for a pattern in a string
- `re.findall(pat, str)`: find all occurrences of a pattern
- `re.fullmatch(pat, str)`: check if entire string matches pattern



- `re.sub(pat, repl, str)`: replace a pattern in a string
- `re.split(pat, str)`: split a string by a pattern

...

#### Note

As always, there is more. But these are a good foundation to build upon.

## Regular Expression in Action

Task: Replace all occurrences of Python by "SECRET".

```
import re
string = """
Python is a programming language.
Python is also a snake.
Monty Python was a theater group.
"""
# Your code here
```

...

#### Note

Regular expressions are even **more powerful** when combined with special characters.

## Special Characters I

- `.` matches any character
- `*` matches zero or more of the preceding element
- `+` matches one or more of the preceding element
- `?` matches zero or one of the preceding element
- `[]` matches any character in the brackets
- `|` matches either the left or the right side
- `\d` matches any digit
- `\w` matches any word character (alphanumeric and underscore)
- `\s` matches any whitespace character

## Special Characters II

- There are **many more special characters** in regular expressions
- In order to keep things simple, we will not cover them here

...

#### Tip

It can be quite complicated to get the hang of these special characters, especially at the beginning. Gladly, there are tools like [regex.com](https://regex.com) that can help with building the right pattern. Apart from that, `help(re)` in the terminal can also be very helpful.

## Advanced Regular Expressions in Action

Task: Use regular expressions to extract all dates from the text.

```
dates = """
On 07-04-1776, the United States declared its independence. Many years later,
on 11-09-1989, the Berlin Wall fell. In more recent history, the COVID-19
pandemic was declared a global emergency on 04-11-2020.
"""

# Try to find all dates in the above text with findall()
# Your code here
```

# Packages

## What are Packages?

- Packages are essentially **collections of modules**
- They can contain multiple modules, subpackages, and data files
- Many packages are available in the Python Package Index (PyPI)
- You don't have to invent the wheel yourself
- **A lot of functionality** is already implemented by others!

## Installing Packages

- Packages are **installed in the shell**
- Use `pip install <package_name>` to install a specific package
- Afterward you can import from the package in your Python scripts

...

### Tip

With Thonny you can install packages directly in the IDE. Simply click on Tools -> Manage packages and search for the package you want to install.

## Virtual Environments

- Virtual environments are used to manage dependencies
- They allow you to have different environments for projects
- They can be created using the `venv` module
- This becomes important if you work on several projects at once

...

### Note

#### **And that's it for today's lecture!**

We now have completed the first step into data science in Python. Next week, we can use this new knowledge to start to work with some tabular data and matrices.

# Literature

## Interesting Books

- Downey, A. B. (2024). Think Python: How to think like a computer scientist (Third edition). O'Reilly. [Link to free online version](#)
- Elter, S. (2021). Schrödinger programmiert Python: Das etwas andere Fachbuch (1. Auflage). Rheinwerk Verlag.

...

### Tip

Nothing new here, but these are still great books!

...

For more interesting literature to learn more about Python, take a look at the [literature list](#) of this course.