Lecture VI - Using Modules and Random Numbers

Programming with Python

Dr. Tobias Vlćek

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 if 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

What are Modules?

- Modules are files containing Python code
- They can define functions, classes, and variables
- They can be imported into other Python scripts
- They can be used to organize code and make it reusable

Creating Modules

- Create a new file with a `.py` extension
- Define functions, classes, and variables in the file
- Import the module in other Python scripts using the `import` statement

Importing Modules

- Use the `import` statement to import the module
- Use the `from` statement to import specific functions, classes, or variables

. . .

```
import my_module
from my_module import my_function
```

Built-in Modules

- Python comes with many built-in modules
- Some common ones: `math`, `random`, `datetime`, `os`, `sys`, `json`, `csv`, `pandas`, `re`

Virtual Environments

- Virtual environments are used to manage dependencies and packages
- They allow you to have different environments for different projects
- They can be created using the `venv` module

Creating a Virtual Environment

- Use the `venv` module to create a virtual environment
- Use the `python -m venv` command to create a virtual environment

. . .

```
python -m venv my_env
```

Activating a Virtual Environment

- Use the `source` command to activate the virtual environment
- Use the `my_env\Scripts\activate` command to activate the virtual environment

. . .

```
source my_env/Scripts/activate
```

Importing Packages

Installing Packages

- Use the `pip install <package_name>` command to install a specific package
- Use the `pip install -r requirements.txt` command to install packages from a requirements file

Importing Packages

- Use the `import <package_name>` statement to import a specific package
- Use the `from <package_name> import <function_name>` statement to import a specific function from a package

Using Modules

Importing Modules

- Use the `import <module_name>` statement to import a specific module
- Use the `from <module_name> import <function_name>` statement to import a specific function from a module

Importing Modules from a Package

- Use the `import <package_name>.<module_name>` statement to import a specific module from a package

Standard Libraries

Random Numbers

- The `random` module provides functions to generate random numbers
- Use the `random.randint(a, b)` function to generate a random integer between `a` and `b`
- Use the `random.choice(list)` function to generate a random element from a list

Math Module

- The `math` module provides functions to perform mathematical operations
- Use the `math.sqrt(x)` function to calculate the square root of `x`
- Use the `math.sin(x)` function to calculate the sine of `x`
- Use the `math.cos(x)` function to calculate the cosine of `x`
- Use the `math.tan(x)` function to calculate the tangent of `x`

OS Module

- The `os` module provides functions to interact with the operating system
- Use the `os.listdir(path)` function to list all files and directories in a directory
- Use the `os.path.join(path, filename)` function to join a path and a filename
- Use the `os.path.exists(path)` function to check if a path exists
- Use the `os.path.isfile(path)` function to check if a path is a file
- Use the `os.path.isdir(path)` function to check if a path is a directory

CSV Module

- CSV (Comma-Separated Values) files are a common format for storing tabular data
- Use the `csv` module to read and write CSV files
- Basic operations:
  - Read: `csv.reader(file)`
  - Write: `csv.writer(file)`

Regular Expressions

What are Regular Expressions?

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

Using Regular Expressions

- Use the `re.search(pattern, string)` function to search for a pattern in a string
- Use the `re.findall(pattern, string)` function to find all occurrences of a pattern in a string
- Use the `re.sub(pattern, replacement, string)` function to replace a pattern in a string
- Use the `re.split(pattern, string)` function to split a string by a pattern

Regular Expression Syntax

- Use the `re.compile(pattern)` function to compile a regular expression
- Use the `re.match(pattern, string)` function to match a pattern at the beginning of a string
- Use the `re.search(pattern, string)` function to search for a pattern in a string
- Use the `re.findall(pattern, string)` function to find all occurrences of a pattern in a string
- Use the `re.sub(pattern, replacement, string)` function to replace a pattern in a string
- Use the `re.split(pattern, string)` function to split a string by a pattern