Python Class Notes

Clarusway

Exception Handling, Decorators, Libraries, pip

Nice to have VSCode Extentions:

- Jupyter

Needs

- Python (for Windows OS: add to the path while installing!)

Summary

- Exception Handling
  - Many Exceptions
  - Raise an exception
- Decorators
  - A Basic Decorator
- Third-party libraries and frameworks
- pip

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

- The try block lets you test a block of code for errors.
- The except block lets you handle the error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.

```
print(x)
# The try block will generate an exception, because the variable is not defined,
to handle that situation;

try:
  print(x)
except:
  print("An exception occurred")
# Since the try block raises an error, the except block will be executed.
```

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error.

```
try:
  print(unknown_var)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")

# It's also possible to print out the actual exception code
# to better inform the user:
try:
  print(unknown_var)
except NameError as e:
  print("Variable is not defined")
  print(e)
except:
  print("Something else went wrong")


# You can use the else keyword to define a block of code
# to be executed if no errors were raised:
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")

# The finally block, if specified, will be executed
# regardless if the try block raises an error or not.
# Generally it is used to close opened files or
# a database connection.
```

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs. This type of error occurs whenever `syntactically correct Python code` results in an error. To throw (or raise) an exception, use the `raise` keyword.

```
# Raise an error and stop the program if x is lower than 0:

x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")

# You can define what kind of error to raise, and the text to print to the user.
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Decorators

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure.

Decorator takes another function as an argument, extending the behavior of that function without explicitly modifying it.

Decorators are usually called before the definition of a function you want to decorate.

Decorators dynamically alter the functionality of a function, method or class without having to make subclasses or change the source code of the decorated class.

Thanks to this our code will be more cleaner, more readable, maintainable (Which is no small thing), and reduce the boilerplate code allowing us to add functionality to multiple classes using a single method.

A Basic Decorator

```
def my_decorator(func):
    def wrapper():
        # Do something before
        result = func()  # or only func()
```

```
        # Do something after
        return result  # or erase this line, with only func() above
    return wrapper

@my_decorator
def func():
    pass
```

Here is a simple decorator example:

```
def make_posh(func):
    def wrapper():
        print('+---------+')
        print('|         |')
        result = func()
        print(result)
        print('|         |')
        print('+=========+')
        return result
    return wrapper

@make_posh
def pfib():
    '''Print out Fibonacci'''
    return ' Fibonacci '
```

Here is another example:

```
import time

def elapsed_time(func):
    def wrapper():
        t1 = time.time()
        func()
        t2 = time.time()
        print(f'Elapsed time: {(t2 - t1) * 1000} ms')
    return wrapper

@elapsed_time
def big_sum():
    num_list = []
    for num in (range(0, 100)):
        num_list.append(num)
    print(f'Big sum: {sum(num_list)}')

big_sum()
```

```
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request)
    pass

# Each time that a user try to access to my_view, the code inside login_required
will be executed.

# A simple view – A view function, or view for short, is a Python function that
takes a Web request and returns a Web response.
```

Third-party libraries and frameworks

Python has a vast ecosystem of third-party libraries and frameworks that can be used for various purposes. Here are some of the most commonly used libraries in Python:

- **NumPy** - a library for working with numerical data in Python, providing support for large, multi-dimensional arrays and matrices, along with a variety of mathematical operations.
- **Pandas** - a library for data manipulation and analysis, providing tools for reading and writing data in a variety of formats, filtering, grouping, and transforming data, and more.
- **Matplotlib** - a library for creating data visualizations in Python, providing a wide range of charts, graphs, and plots.
- **TensorFlow** - an open-source machine learning framework developed by Google, used for building and training deep neural networks.
- **Scikit-learn** - a machine learning library for Python, providing tools for data preprocessing, feature selection, model training, and model evaluation.
- `Flask` - a `lightweight web framework` for building web applications in Python, providing support for request routing, HTTP requests and responses, and more.
- `Django` - a popular web framework for building scalable and robust web applications in Python, providing a `full-stack framework` for web development.
- `Requests` - a library for `making HTTP requests` in Python, providing support for sending GET, POST, PUT, DELETE requests, handling cookies, and more.
- `Beautiful Soup` - a library for `web scraping` in Python, providing tools for parsing HTML and XML documents, and extracting information from them.
- **Pygame** - a library for creating games and multimedia applications in Python, providing support for graphics, sound, input handling, and more.

Note that this is not an exhaustive list, and there are many other libraries and frameworks that can be used in Python, depending on the specific use case.

pip

`pip` is the standard package manager for Python. It allows you to install and manage additional packages that are not part of the Python standard library.

`pip` is already installed if you are using Python 2 >=2.7.9 or Python 3 >=3.4 downloaded from python.org or if you are working in a Virtual Environment created by virtualenv or venv. Just make sure to upgrade pip.

https://pip.pypa.io/en/stable/installing/

```
# Check if pip is installed or not.
pip --version

# See the commands available with pip
pip help

# Upgrade pip
python -m pip install --upgrade pip

# Install any package with pip
pip install <package name>
pip install Django
# https://pypi.org/project/python-decouple/


# See if it is installed or not:
pip list  # List installed packages. or:
pip freeze  # Output installed packages in requirements format.


# Send packages needed in the project to a specific file
pip freeze > requirements.txt


# Install all needed packages with pip
pip install -r requirements.txt
```

☺ **Happy Coding!** ✍

Clarusway