esade

Advanced Programming in Python

LECTURE 3

Do Good. Do Better.

esade

Previously On...

Do Good. Do Better.

Previously On...

Encapsulation

Variables

Specifications

Class

Architecture

Namespaces

Methods

Objects

esade

Introduction

Do Good. Do Better.

Outline

Previously on...
Introduction
Licensing & FOSS
Documentation
Code Modularity
Code Stability
Source Control

esade

Licensing & FOSS

Do Good. Do Better.

Intellectual Property & Copyright

"*Intellectual property (IP) refers to creations of the mind, such as inventions; literary and artistic works; designs; and symbols, names and images used in commerce.*" (WIPO[1])

"*Copyright (or author's right) is a legal term used to describe the rights that creators have over their literary and artistic works.*" (WIPO[2])

1. Retrieved from, World International Property Organization (WIPO), April 2022, https://www.wipo.int/about-ip/en/ (Accessed: 20-04-2022)

2. Retrieved from, World International Property Organization (WIPO), April 2022, https://www.wipo.int/copyright/en/ (Accessed: 20-04-2022)

FOSS & Privative Software

- **FOSS: Free and Open Source Software (4 freedoms)**
  - Freedom to run the software as you wish
  - Freedom to study how the code works (open source)
  - Freedom to redistribute copies
  - Freedom to modify the code
- **Privative Software**
  - Cannot be modified by the user
  - Source code is not accessible
  - Cannot be redistributed without consent from the Author/s

GNU, April 2022, https://www.gnu.org/philosophy/free-sw.en.html (Accessed: 20-04-2022)

FOSS: Licenses

Choosing the right license for you:

- Strong Copyleft (GPL v3):
  - https://choosealicense.com/licenses/gpl-3.0/
- More permisive (Apache v2):
  - https://choosealicense.com/licenses/apache-2.0/
- Short and simple (MIT):
  - https://choosealicense.com/licenses/mit/

esade

Documentation

Do Good. Do Better.

Documentation

```
class session3:
    '''This is a documentation block regarding the
    class 3.'''
    def __init__(self, arg1: int, arg2: str) -> None:
        '''Initialization function
            arg1: it is an int
            arg2: it is a string
            return: None
        '''
        message = "Class with two arguments: {arg1}, {arg2}"
        print(message.format(arg1=arg1, arg2=arg2))

if __name__ == "__main__":
    s = session3(1, 'a')
    print(s.__doc__)
    print(s.__init__.__doc__)
```

Documentation

Global Header

```
## session3.py
#  Documentation and copyright header
#
#  Other Parameters

class session3:
    ...
```

Get Docs

```
>>> import session3
>>> help(session3)
```

Other Packages

- Doxygen
- Sphynx
- Markdown

esade

Code Modularity

Do Good. Do Better.

Code Modularity

```python
class Fruit:
    def __init__(self, size: int, name: int) -> None:
        self.size = size
        self.name = name

    def set_size(self, size: int = 10) -> None:
        self.size = size

if __name__ == "__main__":
    c1 = Fruit(10, "orange")
```

Code Modularity: Arguments

```python
import sys
class Fruit:
    def __init__(self, size: int, name: str) -> None:
        self.size = size
        self.name = name

    def set_size(self, size: int = 10) -> None:
        self.size = size

if __name__ == "__main__":
    c1 = Fruit(sys.argv[1], sys.argv[2])
```

Code Modularity: Argparse

```python
import argparse
class session3:
    def __init__(self, arg1: int, arg2: str) -> None:
        message = "Class with two arguments: {arg1}, {arg2}"
        print(message.format(arg1=arg1, arg2=arg2))

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="This program takes two arguments")
    parser.add_argument('-arg1', '-a1',
        default=0, type=int, required=False)
    parser.add_argument('-arg2', '-a2',
        type=str, required=True, help='argument 2')
    argument_list = parser.parse_args()
    s = session3(argument_list.arg1, argument_list.arg2)
```

esade

Code Stability

Exception control & Test

Do Good. Do Better.

Exceptions

Exceptions are used to:

- Prevent sudden crashes.
- Prevent erroneous behavior of our code.
- Prevents errors in expected failure points.

We make use of the try/except/rise statements in python, accompanied of the library/class we expect to fail.

Asserts (only in non production)

Asserts are used to:

- Check unintended behaviors.
- Limit execution paths.
- Prevent unintended use of the code.

We make use of the assert statement in python. Asserts are typically binary operators that come from a comparison. They will trigger if set to False.

DEMO

Unit Tests

Testing is essential as:

- Prevents errors due to misuse.
- Improves the stability of the code.
- Allows faster debugging of unintended behavior.

We make use of the 'unittest' module from python

```
import unittest
```

Unit Tests: testing the code

Types of test on Objects:

assertEqual(a, b):	a == b
assertTrue(a):	bool(a) is True
assertIs(a, b):	a is b
assertIn(a, b):	a in b
...	

Types of test on Exceptions:

assertRises(exc, fun)

assertWarns(warn, fun)

assertLogs(logger, level)

...

Python Org Unittest, April 2022, https://docs.python.org/3/library/unittest.html (Accessed: 20-04-2022)

Unit Tests: testing the code

session3.py

```python
class session3:
    '''This is a documentation block regarding the
    class 3.'''
    def __init__(self) -> None:
        '''Initialization function
        '''

    def function_1(self, arg1: int, arg2: int) -> int:
        '''Function that does something'''
        return arg1 + arg2

if __name__ == "__main__":
    s = session3()
```

test_session3.py

```python
import unittest
from session3 import session3

class testSession3(unittest.TestCase):
    def setUp(self):
        self.cls = session3()

    def test_int(self):
        self.assertEqual(
            self.cls.function_1(1,1), 2)

if __name__ == '__main__':
    unittest.main()
```

To launch the test: `python -m unittest test_session3.py`

esade

DEMO

esade

Source Control

Do Good. Do Better.

Source control: Git

Version control is essential as:

- Allows better control of the changes made into the code
- Allows for better collaborative environment
- Helps track modifications through time
- Allows efficient handling of large projects

Source control: Git

Important commands

Initialize repository:	`git init`
Check resporitory status:	`git status`
Add files to the repository:	`git add <file>`
Commit files:	`git commit -m 'message'`

Source control: Contributing

Important commands for contributing

Clone repository:	`git clone <url>`
Pull repository:	`git pull origin master`
Push changes to repository:	`git push origin master`
Checkout branch:	`git checkot -b <branch>`
Stash changes:	`git stash <files>`

DEMO

esade

Do Good. Do Better.