Python Programming Language Foundation

Session 5

Session overview

- Closures
- Decorators
- Files
- Modules

Closures

Definition

```
from time import sleep

def time_printer(sec):
    def printer(msg):
        sleep(sec)
        print(msg)

    return printer
```

Definition

```
>>> printer5 = time_printer(5)
>>> printer5('Hello')
'Hello'   # After 5 sec
```

Definition

A **Closure** is a **function** object that **remembers values** in **enclosing scopes** even after function finished working

https://www.learnpython.org/en/Closures

Using

Nested functions

Decorators

Decorators

Pattern

```python
import time

def timer(f):
    def wrapper(*args, **kwargs):
        t0 = time.time()
        res = f(*args, **kwargs)
        t1 = time.time()
        print(f'Function execution time: {t1 - t0:.2f}')
        return res

    return wrapper
```

Pattern

```python
import time

def sleeper(sec):
    time.sleep(sec)

timed_sleeper = timer(sleeper)
timed_sleeper(5)  # Function execution time: 5.00
```

Definition

A **decorator** is a **function** that takes another **function** and extends the **behavior** of the latter function without explicitly **modifying** it

https://realpython.com/primer-on-python-decorators

Syntax sugar

```
import time

@timer   # Syntax sugar
def sleeper(sec):
    time.sleep(sec)

sleeper(5)   #Function execution time: 5.00
```

Decorator parametrization

```python
import time

def paused(t):
    def parametrized_paused(f):
        def wrapper(*args, **kwargs):
            time.sleep(t)
            return f(*args, **kwargs)

        return wrapper

    return parametrized_paused
```

Decorator parametrization

```python
@paused(4)
def func(x, y):
    return x + y
```

Decorators usage

- Function calls tracing
- Pre- and post- conditions setup
- Synchronization
- Lazy evaluation and function call results caching
- Tail recursion optimization
- Function arguments type checking
- Etc.

Files

Open / close files

```
f = open('file_name.txt')
file_content = f.read()
f.close()
```

`open` function

```
open(
    file,
    mode='r',
    buffering=-1,
    encoding=None,
    errors=None,
    newline=None,
    closefd=True,
    opener=None,
)
```

https://docs.python.org/3.9/library/functions.html#open

File modes

Mode	Description
'r'	Read-only (is used by default).
'w'	Write-only. If there is no file with such a name it will be created. If the file exists it’s contents will be overwritten.
'x'	The same as ‘w’ but raises an exception if the file already exists.
'a'	Appending.
'b'	Binary mode.
't'	Text mode (is used by default).
'+'	Read-write.

File methods

file	close()
	read(size=-1)
	readline(size=-1)
	readlines(hint=-1)
	write(b)
	writelines(lines)
	writable()
	seek(offset, whence=SEEK_SET)
	tell()

Modules

Loading the module’s contents with `import`

Module `greeter.py`

```python
def greet(person):
    print(f'Hello, {person}!')
```

Some other module

```python
# Import module greeter
import greeter

# Call the module's function
greeter.greet('EPAM Student')
```

Loading the module’s contents with `import as`

Module `greeter.py`

```
def greet(person):
    print(f'Hello, {person}!')
```

Some other module

```
# Import module greeter
import greeter as hello

# Call the module's function
hello.greet('EPAM Student')
```

Loading the module’s contents with `from … import`

Module `greeter.py`

```
def greet(person):
    print(f'Hello, {person}!')
```

Some other module

```
# Import from module greeter
from greeter import greet

# Call the module's function
greet('EPAM Student')
```

Loading the module’s contents with `from … import *`

Module `greeter.py`

```
def greet(person):
    print(f'Hello, {person}!')


def learn_pl(pl):
    print(f'{pl} was learned!')
```

Some other module

```
# Import from module greeter
from greeter import *

# Call the module's function
greet('EPAM Student')
learn_pl('Python')
```

Python module search path

?

The directory containing the input script

PYTHONPATH

The installation-dependent default	`/usr/local/lib/python`
	`C:\pythonXX\lib`
	`sys.path`

Built-in modules (standard library)

Packages

Packages

Package structure

```
# pip install tree

$ tree excellent_engineer

excellent_engineer
├── __init__.py
├── __main__.py
├── skill_c.py
├── skill_nim.py
├── skill_linux.py
└── skill_python.py
```

`<SKILL-NAME>.py` module example

```python
def <SKILL-NAME>():
    print('<SKILL-NAME>')
```

Packages

Module `excellent_engineer/__init__.py`

```
from .skill_python import python
from .skill_c import c
from .skill_nim import nim
from .skill_linux import linux
```

Some other module

```
import excellent_engineer

print(dir(excellent_engineer))

excellent_engineer.python()
excellent_engineer.c()
excellent_engineer.nim()
excellent_engineer.linux()
```

Packages

Module `excellent_engineer/__main__.py`

```python
from .skill_python import python
from .skill_c import c
from .skill_nim import nim
from .skill_linux import linux

def learn_all():
    python()
    c()
    nim()
    linux()

if __name__ == '__main__':
    learn_all()
```

Execute `excellent_engineer`

```
$ python -m excellent_engineer
# Dots should be deleted
$ python excellent_engineer
```

Standard libraries

Libraries

- math/random
- collections
- functools/itertools
- os/sys
- json/csv/yaml
- time/datetime
- logging/argparse

https://docs.python.org/3/library/index.html

Thanks for attention

Questions?