Python | Main course

Session 18 & 19

- Python Interactive shell
- Python Scripts
- argparse
- Python Date & Time

by Mohammad Amin H.B. Tehrani

www.maktabsharif.ir

interactive shell

Intro

The interactive shell is between the user and the operating system (e.g. Linux, Unix, Windows or others). Instead of an operating system an interpreter can be used for a programming language like Python as well. The Python interpreter can be used from an interactive shell.

The interactive shell is also interactive in the way that it stands between the commands or actions and their execution. In other words, the shell waits for commands from the user, which it executes and returns the result of the execution. Afterwards, the shell waits for the next input.

Run:

```
python ...
python3 ....
```

Interactive shell

Example

```
m-tehrani@MohammadAmin:~$ python -3
Python 2.7.18 (default, Mar  8 2021, 13:02:45)
[GCC 9.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> s = 'Hello world!'
>>> s
'Hello world!'
>>> import os
>>> os.listdir()
['.fonts', 'Downloads', '.android', 'VirtualBox VMs', '.xdman', '.ssh', 'javasharedresources',
"PlayOnLinux's virtual drives", 'Public', 'Desktop', '.local', '.xdm-global-lock', '.ssr', 'Matlab',
'.matlab', '.java', 'PycharmProjects', '.PlayOnLinux', 'Templates', '.pki', '.profile', '.thunderbird',
'Pictures', '.python_history', 'snap', '.gnome', '.config', '.bash_logout', '.mozilla', '.bashrc', 'Music',
'Documents', '.cache', 'Videos', '.bash_history', '.gnupg', '.sudo_as_admin_successful']
```

Python Scripts

Intro

Scripting languages do not require the compilation step and are rather interpreted.

Python is **scripting**, general-purpose, high-level, and interpreted programming language. It also provides the object-oriented programming approach.

Run python scripts:
`python <python file with .py>`

```
# test.py
from os import listdir

print(listdir())
```

```
m-tehrani@MohammadAmin:~$ python test.py
['docstring_test.py', 'app.py', 'static',
'__pycache__', 'auth', 'menu', 'venv', '.idea',
'templates']
```

___name___

The **__name__** variable (two underscores before and after) is a special Python variable. It gets its value depending on how we execute the containing script. you can import that script as a module in another script.
When you run your script, the **__name__** variable equals **__main__**. When you import the containing script, it will contain the name of the script.

```
# a_module.py

print('Inside a_module.py, name:', __name__)
```

```
# test.py
import a_module

print('Inside test.py, name:', __name__)
```

Run test.py:

```
m-tehrani@MohammadAmin:~$ python3 test.py
Inside a_module.py, name: a_module
Inside test.py, name: __main__
```

Run a_module.py:

```
m-tehrani@MohammadAmin:~$ python3 a_module.py
Inside a_module.py, name: __main__
```

Scripts

___name___ == '___main___'

We can use an **if __name__ == "__main__"** block to allow or prevent parts of code from being run when the modules are imported. When the Python interpreter reads a file, the **__name__** variable is set as **__main__** if the module being run, or as the module's name if it is imported.

```
# a_module.py

print('Inside a_module.py, name:', __name__)

if __name__ == '__main__':
   print('You can see me if you run me!!!')
```

```
# test.py
import a_module

print('Inside test.py, name:', __name__)
```

Run test.py:

```
m-tehrani@MohammadAmin:~$ python3 test.py
Inside a_module.py, name: a_module
Inside test.py, name: __main__
```

Run a_module.py:

```
m-tehrani@MohammadAmin:~$ python3 a_module.py
Inside a_module.py, name: __main__
You can see me if you run me!!!
```

Script

Example: Screen shot

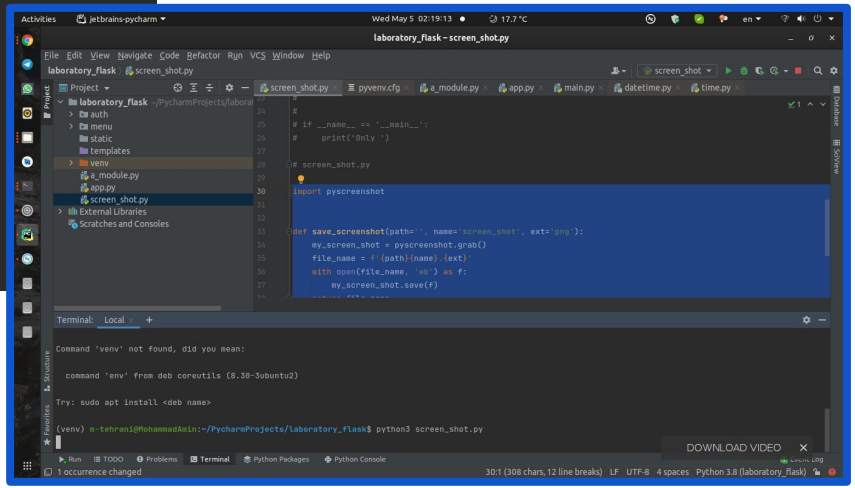
```
import pyscreenshot


def save_screenshot(path='', name='screen_shot',
ext='png'):
    my_screen_shot = pyscreenshot.grab()
    file_name = f'{path}{name}.{ext}'
    with open(file_name, 'wb') as f:
        my_screen_shot.save(f)
    return file_name


if __name__ == '__main__':
    print(save_screenshot())
```

```
m-tehrani@MohammadAmin:~$ python3 screen_shot.py
screen_shot.png
```

Result!



Scripts

python -m ...

You can run library module as a script by using **-m** option.
Now you can run every accessible modules. (from your directory)

Example:

```
m-tehrani@MohammadAmin:~$ python3 -m pip install test
```

```
m-tehrani@MohammadAmin:~$ python3 -m py_compile test.py
```

Exercise: Directory size script

Maktab Sharif

Directory size script

Create a function (get_dir_size for example) that gets a directory path, Then returns the total size of subdirectories and files inside that.
Then implement main condition for the .py file that prints the size of directory that called from.

- Use os.scandir to iterate on directories and files inside the parent directory.
- Create a **decorator "convert_unit"** to convert the function output to a special unit like 'kb', 'mb', ...
- Get current directory: **os.getcwd()**

```
def convert unit(unit: Literal['B', 'KB', 'MB', 'GB']):
    "Decorator: Returns converted result from byte"
    # TODO: Complete here


@convert unit('KB')
def get directory size(directory: str) -> int:
    """Returns the `directory` size in bytes."""
    # TODO: Complete here

if   name   == '  main  ':
    # TODO: Complete here
    pass
```

argparse

Intro

Your program will accept an arbitrary number of **arguments** passed from the command-line (or terminal) while getting executed.
You access them from **sys.argv** in python:

Notice that the **first argument** is always **the name of the Python file** and result is **a list of strings.**

```
import sys

if   name   == '  main__':
   print(sys.argv)
```

```
python3 screen_shot.py --name akbar --sen 12 1 2 3 4
['screen_shot.py', '--name', 'akbar', '--sen', '12', '1', '2', '3', '4']
```

Exercise: Translator again!

Maktab Sharif

Translator script with arguments

Implement a main condition to your translator project, to gets **file_path**, **to_language**, **from_language**, **provider** respectively. Then reads the file and print the result.

- from_language is ‘auto’ by default
- provider is ‘google’ by default

```
m-tehrani@MohammadAmin:~$ python3 test.py test.txt fa auto bing
Using Iran server backend.
سلام اکبر،
چطور هستید؟؟؟
```

argparse module

The **argparse** module makes it easy to write user-friendly command-line interfaces.
The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv.
The argparse module also automatically generates **help** and **usage** messages and **issues** errors when users give the program invalid arguments.

See full document: Argparse tutorial

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Test project')

    parser.add_argument('-i', '--id', metavar='ID', action='store', required=True, help='ID field')
    parser.add_argument('-n', '--name', action='store', default='Akbar', help='Name field')
    ...

    args = parser.parse_args()
    ...
```

argparse

Example: Screenshot with args

Complete source:
Github Repo

```
import pyscreenshot, argparse

def save_screenshot(path, name, ext):
    my_screen_shot = pyscreenshot.grab()
    file_name = f'{path}{name}.{ext}'
    with open(file_name, 'wb') as f:
        my_screen_shot.save(f)
    return file_name


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Take screenshot example')

    parser.add_argument('-p', '--path', action='store', metavar='DIR_PATH', default='',
                        help='directory path for save')
    parser.add_argument('-n', '--name', action='store', metavar='NAME', default='screen_shot',
                        help='file name for save')
    parser.add_argument('-e', '--ext', action='store', metavar='EXT', choices=['png', 'jpg', 'jpeg'], default='png',
                        help='extension of image file')

    args = parser.parse_args()

    file_path = save_screenshot(args.path, args.name, args.ext)
    print('File path:', file_path)
```

Example: Screenshot with arguments

```
m-tehrani@MohammadAmin:~$ python3 screen_shot.py -h
usage: screen_shot.py [-h] [-p DIR_PATH] [-n NAME] [-e EXT]

Take screenshot example

optional arguments:
  -h, --help            show this help message and exit
  -p DIR_PATH, --path DIR_PATH
                        directory path for save
  -n NAME, --name NAME  file name for save
  -e EXT, --ext EXT     extension of image file
```

```
m-tehrani@MohammadAmin:~$ python3 screen_shot.py -n test_sc
File path: test_sc.png
```

```
m-tehrani@MohammadAmin:~$ python3 screen_shot.py -n test_sc --path images/ -e jpg
File path: images/test_sc.jpg
```

Example: User register

Complete source:
Github Repo

```
import argparse

class User: pass


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='User Registration script')

    parser.add_argument(metavar='ID', action='store', dest='id', type=int, help='User id')
    parser.add_argument(metavar='FirstName', action='store', dest='first_name', help='First name')
    parser.add_argument(metavar='LastName', action='store', dest='last_name', help='Last name')
    parser.add_argument('-p', '--phone', metavar='Phone', action='store', required=True, help='Phone number')
    parser.add_argument('-e', '--email', metavar='Email', action='store', default=None, help='Email address')
    parser.add_argument('-a', '--age', metavar='Age', action='store', type=int, default=None)
    parser.add_argument('-g', '--gender', metavar='Gender', action='store', choices=['male', 'female'])
    parser.add_argument('--admin', metavar='Admin', action='store_const', const='admin', default='user', dest='type')
    parser.add_argument('--active', action='store_true', dest='active')
    parser.add_argument('--extra', action='store', nargs='*', help='Extra arguments')

    args = parser.parse_args()
    extra = args.extra or ()
    u = User(id=args.id, first_name=args.first_name, last_name=args.last_name, phone=args.phone, email=args.email,
             age=args.age, gender=args.gender, type=args.type, active=args.active, *extra)
    print(u)
```

argparse

Example: Screenshot with arguments

```
m-tehrani@MohammadAmin:~/Makab52/session18-19$ python3 user_manager.py -h
usage: user_manager.py [-h] -p Phone [-e Email] [-a Age] [-g Gender] [--admin] [--active] [--extra [EXTRA [EXTRA ...]]] ID
FirstName LastName

User Registration script

positional arguments:
  ID                    User id
  FirstName             First name
  LastName              Last name

optional arguments:
  -h, --help            show this help message and exit
  -p Phone, --phone Phone
                        Phone number
  -e Email, --email Email
                        Email address
  -a Age, --age Age
  -g Gender, --gender Gender
  --admin
  --active
  --extra [EXTRA [EXTRA ...]]
                        Extra arguments
```

Exercise: Translator argparse

Maktab Sharif

Translator script with arguments argparse

Implement a main condition to your translator project, and prepare arguments to have a help text like this:

Note:
If file_path passed empty, program must run as shell mode and gets input from user with Continuous input() s, until **KeyboardInterrupt (Ctrl+C)**

```
usage: test.py [-h] -t TO_LANG [-f FROM_LANG] [-p PROVIDER] [-s SAVE_PATH]
[file_path]

Translator project

positional arguments:
  file_path             file path to translate (empty for interactive shell)

optional arguments:
  -h, --help            show this help message and exit
  -t TO_LANG, --to_lang TO_LANG
                        select destination language
  -f FROM_LANG, --from_lang FROM_LANG
                        select origin language
  -p PROVIDER, --provider PROVIDER
                        select provider
  -s SAVE_PATH, --save SAVE_PATH
                        select a path to save the result (optional)
```

Python Date & Time

Date & Time

Intro

A Python program can handle date and time in several ways. Converting between date formats is a common chore for computers. Python's time and calendar modules help track dates and times.

Epoch

The epoch, then, is the starting point against which you can measure the passage of time.

Time intervals are floating-point numbers in units of seconds. Particular instants in time are expressed in seconds since 00:00:00 hrs January 1, 1970(epoch).

```
import time  # This is required to include time module.

ticks = time.time()
print("Number of ticks since 12:00am, January 1, 1970:" ticks)
```

```
Number of ticks since 12:00am, January 1, 1970: 1620207996.7637713
```

Decimal figures???

Date & Time

Time

Most important functions

- time.**time()** : Return the time in seconds since the epoch as a floating point number.
- time.**time_ns()** : Similar to time() but returns time as an integer number of nanoseconds since the epoch. (new at v 3.7)
- time.**struct_time()** : It is an object with a named tuple interface: values can be accessed by index and by attribute name.
- time.**gmtime()** : Convert a time expressed in seconds since the epoch to a struct_time in UTC in which the dst flag is always zero. If secs is not provided or None, the current time as returned by time() is used.
- time.**sleep()** : Suspend execution of the calling thread for the given number of seconds. The argument may be a floating point number to indicate a more precise sleep time.
- time.**strptime()** : Parse a string representing a time according to a format. The return value is a struct_time
- time.**strftime()** : Convert a tuple or struct_time representing a time as returned by gmtime() or localtime() to a string as specified by the format argument. If t is not provided, the current time as returned by localtime() is used.

```
import time

birthday = input("Enter your birthday like this: YYYY-MM-DD\n")
struct_t = time.strptime(birthday, '%Y-%m-%d')
print(time.strftime('%a, %B %d, %Y', struct_t))
doy = time.strftime('%j', struct_t)
print(f"{365-int(doy)} days left until your birthday!")
```

Date & Time

Datetime module

The **datetime** module supplies **classes** for manipulating dates and times.

- time
- date
- datetime
- timedelta
- ...

```
from datetime import time, timedelta, date, datetime

t = time(hour=10, minute=10, microsecond=100)  # 10:10:00.100
d = date(year=2001, month=2, day=2)  # 2001-02-02
dt = datetime(year=2002, month=2, day=2, hour=10, minute=2)  # 2002-02-02 10:02:0.000

print(t, d, dt, sep='\n')
print(dt - timedelta(days=31, minutes=10))
```

Date & Time

Datetime module

The **datetime** module supplies **classes** for manipulating dates and times.

- time
- date
- datetime
- timedelta
- ...

```
from datetime import time, timedelta, date, datetime

t = time(hour=10, minute=10, microsecond=100)  # 10:10:00.100
d = date(year=2001, month=2, day=2)  # 2001-02-02
dt = datetime(year=2002, month=2, day=2, hour=10, minute=2)  # 2002-02-02 10:02:0.000

print(t, d, dt, sep='\n')
print(dt - timedelta(days=31, minutes=10))
```

Static & Class methods

```
from datetime import time, date, datetime

# Static & Class methods
t1 = time.fromisoformat('10:10:20')

d1 = date.today()
d2 = date.fromisoformat("1881-10-25")
d3 = date.fromisocalendar(2020, 4, 2)
d4 = date.fromtimestamp(178766776.222)

dt1 = datetime.today()
dt2 = datetime.fromisoformat("1881-10-25")
dt3 = datetime.fromisocalendar(2020, 4, 2)
dt4 = datetime.fromtimestamp(178766776.222)

print('Times:')
print(t1, sep='\n')
print('\nDates:')
print(d1, d2, d3, d4, sep='\n')
print('\nDatetimes:')
print(dt1, dt2, dt3, dt4, sep='\n')
```

```
Times:
10:10:20

Dates:
2021-05-05
1881-10-25
2020-01-21
1975-09-01

Datetimes:
2021-05-05 20:29:44.139804
1881-10-25 00:00:00
2020-01-21 00:00:00
1975-09-01 04:56:16.222000
```

Example: Stopwatch

```python
from datetime import datetime
from time import sleep

if __name__ == '__main__':
    t0 = datetime.now()
    try:
        while True:
            t = datetime.now()
            print(t-t0)
            sleep(1)
    except KeyboardInterrupt:
        print("\nFinished:", datetime.now() - t0)
```

```
m-tehrani@MohammadAmin:~$ python3 stopwatch.py
0:00:00.000005
0:00:01.000764
0:00:02.001832
0:00:03.003020
0:00:04.004300
0:00:05.005606
0:00:06.006514
0:00:07.007772
0:00:08.009033
^C
Finished: 0:00:08.731660
```

Exercise: CountDown script

Maktab Sharif

Countdown script

Write a countdown timer with 2 major functionalities below:

1. Gets a positional time argument **(target_time)**, and start a countdown timer to reach that.
2. Gets **hour, minute, sec** options to set **target_time = now + timedelta** and ...

-> Print the timer values (hour, min, sec) every 1 second until the end.

Hint:

- Use time.sleep() to suspend program (1 second)
- Be on the notice for **KeyboardInterrupt** (Ctrl + C) to break the program.

```
m-tehrani@MohammadAmin:~$ python3 countdown.py 22:05:48
22:05:44
22:05:45
22:05:46
22:05:47
22:05:48
Times UP!!!
```

```
m-tehrani@MohammadAmin:~$ python3 countdown.py -s 5
22:05:44
22:05:45
22:05:46
22:05:47
22:05:48
Times UP!!!
```

Exercise: Translator

Maktab Sharif

Install ‘translators’ package using pip and create a decorator that translate output of the function to a target language.

A) Translate decorator with no parameter:

- Install and Use **‘translators’** package
- Use a desired provider like google
- Assume from_language = ‘auto’
- Assume to_language = ‘fa’

B) Translate decorator with parameters

- Get provider, from_lang, to_lang from parameters
- See: Decorator with parameters article

C) Translate decorator class:

- See: Decorator Class article for Decorators as class
- Re-write Parts A and B using Decorator class

```
# Prototypes
# decorator without parameters
def translator(func):

    def inner(*args, **kwargs):
        ...  # TODO
    return inner

# decorator with parameters
def translator(
        to_lang,
        from_lang='auto',
        provider='google'):

    def inner(func):
        ...  # TODO
    return inner
```

Advanced topics

- Timezone
- Jdatetime
- Venv (Virtual Environment)