Python | Main course

Session 12

- Argparse
- Date & Time
- Venv (Virtual Environment)

by Mohammad Amin H.B. Tehrani - Reza Yazdani

www.maktabsharif.ir

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)
```

```
python script.py --name akbar --age 12 1 2 3 4
['script.py', '--name', 'akbar', '--age', '12', '1', '2', '3', '4']
```

Setting up a Parser

The first step when using argparse is to create a parser object and tell it what arguments to expect. The parser can then be used to process the command line arguments when your program runs.

The parser class is **ArgumentParser.** The constructor takes several arguments to set up the description used in the help text for the program and other global behaviors or settings.

```
import argparse


parser = argparse.ArgumentParser(description='This is a python 78 sample script')
```

Defining Arguments

argparse is a complete argument processing library. Arguments can trigger different actions, specified by the action argument to **add_argument()**. Supported actions include storing the argument (singly, or as part of a list), storing a constant value when the argument is encountered (including special handling for true/false values for boolean switches), counting the number of times an argument is seen, and calling a callback.

The default action is to store the argument value. In this case, if a type is provided, the value is converted to that type before it is stored. If the dest argument is provided, the value is saved to an attribute of that name on the Namespace object returned when the command line arguments are parsed.

argparse

Simple Examples

Here is a simple example with 3 different options:
a boolean option (**-b**), a simple string option (**-s**), and an integer option (**-i**).

```
parser = argparse.ArgumentParser(description='Short sample script')

parser.add_argument('-b', action="store_true", default=False)
parser.add_argument('-s', action="store", dest="s")
parser.add_argument('-i', action="store", dest="i", type=int)

print(parser.parse_args(['-b', '-svalue', '-i', '3']))
print(parser.parse_args(['-s', 'value']))
```

```
┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
└─λ python script.py
Namespace(b=True, s='value', i=3)
Namespace(b=False, s='value', i=None)
```

Simple Examples

There are a few ways to pass values to single character options. The example above uses two different forms, **-cval** and **-c val**.

“Long” option names, with more than a single character in their name, are handled in the same way.

```
parser = argparse.ArgumentParser(description='Example with long option names')

parser.add_argument('--noarg', action="store_true", default=False)
parser.add_argument('--witharg', action="store", dest="witharg")

print(parser.parse_args(['--noarg', '--witharg', 'val']))
```

```
┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 3ms
└─λ python script.py
Namespace(noarg=True, witharg='val')
```

Argument Actions

There are six built-in actions that can be triggered when an argument is encountered:

store: Save the value, after optionally converting it to a different type. This is the default action taken if none is specified explicitly.

store_const : Save a value defined as part of the argument specification, rather than a value that comes from the arguments being parsed. This is typically used to implement command line flags that aren't booleans.

store_true | store_false : Save the appropriate boolean value. These actions are used to implement boolean switches.

append: Save the value to a list. Multiple values are saved if the argument is repeated.

append_const: Save a value defined in the argument specification to a list.

version: Prints version details about the program and then exits.

argparse

Simple Example

```
import argparse

parser = argparse.ArgumentParser()

parser.add_argument('-s', action='store', dest='simple_value', help='Store a simple value')
parser.add_argument('-c', action='store_const', dest='constant_value', const='value-to-store',
                    help='Store a constant value')
parser.add_argument('-t', action='store_true', default=False, dest='boolean_switch', help='Set a switch to true')
parser.add_argument('-f', action='store_false', default=False, dest='boolean_switch', help='Set a switch to false')
parser.add_argument('-a', action='append', dest='collection', default=[], help='Add repeated values to a list', )
parser.add_argument('-A', action='append_const', dest='const_collection', const='value-1-to-append', default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection', const='value-2-to-append',
                    help='Add different values to list')
parser.add_argument('--version', action='version', version='%(prog)s 1.0')   # what's `prog` ? :)

results = parser.parse_args()

print('simple_value     =', results.simple_value)
print('constant_value   =', results.constant_value)
print('boolean_switch   =', results.boolean_switch)
print('collection       =', results.collection)
print('const_collection =', results.const_collection)
```

argparse

Simple Examples

```
 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -h
usage: script.py [-h] [-s SIMPLE_VALUE] [-c] [-t] [-f] [-a COLLECTION] [-A] [-B] [--version]

options:
  -h, --help       show this help message and exit
  -s SIMPLE VALUE  Store a simple value
  -c               Store a constant value
  -t               Set a switch to true
  -f               Set a switch to false
  -a COLLECTION    Add repeated values to a list
  -A               Add different values to list
  -B               Add different values to list
  --version        show program's version number and exit

 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -s value

simple value     = value
constant value   = None
boolean switch   = False
collection       = []
const_collection = []

 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py --version

script.py 1.0
```

argparse

Simple Examples

```
 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -c

simple_value     = None
constant_value   = value-to-store
boolean_switch   = False
collection       = []
const_collection = []

 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -t

simple value     = None
constant_value   = None
boolean_switch   = True
collection       = []
const_collection = []

 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -f

simple value     = None
constant value   = None
boolean_switch   = False
collection       = []
const_collection = []
```

argparse

Simple Examples

```
 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -a one -a two -a three

simple_value     = None
constant_value   = None
boolean_switch   = False
collection       = ['one', 'two', 'three']
const_collection = []

 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -B -A

simple value     = None
constant_value   = None
boolean_switch   = False
collection       = []
const_collection = ['value-2-to-append', 'value-1-to-append']

 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python script.py -A -a hi_maktab_78 -a akbar_say_hello -A

simple value     = None
constant value   = None
boolean_switch   = False
collection       = ['hi_maktab_78', 'akbar_say_hello']
const_collection = ['value-1-to-append', 'value-1-to-append']
```

argparse

Example: Screenshot with args

Complete source:
Github Repo

```
import pathlib
import argparse
import pyscreenshot


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 script')

    parser.add_argument('-p', '--path', action='store', metavar='PATH', default=pathlib.Path(__file__).parent.resolve(),
                        help='directory path for save')
    parser.add_argument('-n', '--name', action='store', metavar='NAME', default='screenshot',
                        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(f'Screenshot saved in {file_path}')
```

Example: Screenshot with arguments

```
 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python screenshot.py -h
usage: screenshot.py [-h] [-p PATH] [-n NAME] [-e EXT]

Take screenshot script

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

```
 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python screenshot.py -n test_name
Screenshot saved in /home/yazdan/Desktop/Python78/test_name.png
```

```
 ┌─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 9ms
 └─λ python screenshot.py -n my_shot -e jpg
Screenshot saved in /home/yazdan/Desktop/Python78/my_shot.jpg
```

argparse

Example: User register

Complete source:
Github Repo

```
import argparse


class User: ... # complete source in github page

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

    parser.add_argument('id', metavar='ID', action='store', type=int, help='User id')  # it's pos arg
    parser.add_argument('first name', metavar='FirstName', action='store', help='First name')  # it's pos arg
    parser.add_argument(metavar='LastName', action='store', dest='last name', help='Last name')  # it's pos arg too
    parser.add_argument('-p', '--phone', metavar='Phone', action='store', required=True,
                        help='Phone number')  # It's required!
    parser.add_argument('-e', '--email', metavar='Email', action='store', default=None,
                        help='Email address')  # It's optional
    parser.add_argument('-a', '--age', metavar='Age', action='store', type=int, default=None)  # Type: int
    parser.add_argument('-g', '--gender', metavar='Gender', action='store', choices=['male', 'female'])  # choices: ...
    parser.add_argument('--admin', metavar='Admin', action='store_const', const='admin', default='user',
                        dest='type')  # This option does Not get value, just stores a const value (admin) into dest (type) if it mentioned
    parser.add_argument('--active', action='store_true',
                        dest='active')  # This option does Not get value, just stores a True value into dest (active) if it mentioned
    parser.add_argument('--extra', action='store', nargs='*',
                        help='Extra arguments')  # nargs = '*' -> Empty or More argument

    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)
```

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 12:00 am, January 1, 1970:", ticks)
```

```
Number of ticks since 12:00 am, January 1, 1970: 1660297226.3882096
```

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

Static & Class methods

```python
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:
2022-08-12
1881-10-25
2020-01-21
1975-09-01

Datetimes:
2022-08-12 14:14:50.834434
1881-10-25 00:00:00
2020-01-21 00:00:00
1975-09-01 04:56:16.222000
```

Date & Time

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)
```

```
╭─yazdan@DarkBook in repo: Python78 via v3.10.5 (venv) took 9ms
╰─λ 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
```

Venv
Virtual Environment

Venv

Intro

The venv module provides support for creating lightweight “virtual environments” with their own site directories, optionally isolated from system site directories.

Each virtual environment has its own Python binary (which matches the version of the binary that was used to create this environment) and can have its own independent set of installed Python packages in its site directories.

Installing

If you are using Python 3.3 or newer, the **venv** module is the preferred way to create and manage virtual environments. venv is included in the Python standard library and requires no additional installation.
If you are using venv, you may skip this section.

virtualenv is used to manage Python packages for different projects. Using virtualenv allows you to avoid installing Python packages globally which could break system tools or other projects.
You can install virtualenv using pip.

$ pip install virtualenv

Venv

Creating a virtual environment

venv (for Python 3) and virtualenv (for Python 2) allow you to manage separate package installations for different projects. They essentially allow you to create a “virtual” isolated Python installation and install packages into that virtual installation. When you switch projects, you can simply create a new virtual environment and not have to worry about breaking the packages installed in the other environments. It is always recommended to use a virtual environment while developing Python applications.

To create a virtual environment, go to your project’s directory and run venv. If you are using Python 2, replace venv with virtualenv in the below commands.

$ python -m venv venv

The second argument is the location to create the virtual environment. Generally, you can just create this in your project and call it venv.

venv will create a virtual Python installation in the venv folder.

Activating a virtual environment

Before you can start installing or using packages in your virtual environment you'll need to activate it. Activating a virtual environment will put the virtual environment-specific python and pip executables into your shell's PATH.

Platform	Shell	Command to activate virtual environment
POSIX	bash/zsh	$ source <venv>/bin/activate
	fish	$ source <venv>/bin/activate.fish
	csh/tcsh	$ source <venv>/bin/activate.csh
Windows	PowerShell Core	$ <venv>/bin/Activate.ps1
	cmd.exe	C:\> <venv>\Scripts\activate.bat
	PowerShell	PS C:\> <venv>\Scripts\Activate.ps1

Leaving the virtual environment

If you want to switch projects or otherwise leave your virtual environment, simply run:

$ deactivate

If you want to re-enter the virtual environment just follow the same instructions above about activating a virtual environment. There's no need to re-create the virtual environment.

Advanced topics

- Timezone
- Jdate
- Jdatetime