Python for videogames

Videogames Technology
Asignatura transversal

Departamento de Automática

Objectives

1. First contact with Arcade.
2. Understand modules and packages.
3. Basic package management.
4. Introduce virtual environments.

Bibliography

- The Python Tutorial. *Chapter 6: Modules.* (Link)
- Paul Vincent Craven. *Easy 2D game creation with Arcade.* (Link)
- Paul Vincent Craven. *Learn to Program with Arcade.* (Link)

Table of Contents

Introduction

Why modules?

- **Main function**: Organization.
- **Reuse**: To provide software solutions, that have been proven to work, to solve similar problems.

Modules

Creation

A module is just a Python script with `.py` extension

fibo.py

```python
def fib(n):
    """Print a Fibonacci series up to n"""
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):
    """Print a Fibonacci series up to n"""
    result = [] # Declare a new list
    a, b = 0, 1
    while a < n:
        result.append(a) # Add to the list
        a, b = b, a+b
    return result
```

Modules

Using modules (I)

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
>>> fib = fibo.fib
>>> fib(100)
1 1 2 3 5 13 21 34 55 89
```

Modules

Using modules (II)

A module can import other modules

- Name conflicts may arise: Each module has a symbol table
- It means you should invoke it as `modname.itemname`

It is possible to import items directly

- `from module import name1, name2`
- `from module import *`
- It uses the global symbol table (no need to use the modname)

```
>>> from fibo import fib, fib2
>>> fib(100)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Modules

Using modules: Example

List zip file contents (file.zip must exist. Open in read mode)

```
import zipfile

file = zipfile.ZipFile("file.zip", "r")

# list filenames
for name in file.namelist():
    print(name)

# list file information
for info in file.infolist():
    print(info.filename, info.date_time, info.file_size)
```

Several examples here: `http://pymotw.com/2/PyMOTW-1.132.pdf`

Modules

Modules as scripts (I)

When a module is imported, its statements are executed

- It declares functions, classes, variables ...
- ... and also executes code
- It serves to initialize the module

Very useful to use modules as programs and libraries

Modules

Modules as scripts (II)

fibo2.py

```python
def fib(n):
    """Print a Fibonacci series up to n"""
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

`__name__` is a special variable set to the module's name

```
   (In Linux console)
$ python3 fibo2.py 50
1 1 2 3 5 8 13 21 34
```

```
    (In Python interpreter)
>>> import fibo2
>>> fibo2.fib(50)
1 1 2 3 5 8 13 21 34
```

Module

The dir() function

`dir()`: Built-in function that returns the names defined in a module

- Without arguments, it returns your names

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fib', 'fib2']
>>> dir()
['__builtins__', ... , '__spec__']
>>> variable = 'Hello'
>>> dir()
['__builtins__', ... , '__spec__', 'variable']
```

Packages

Package concept (I)

If a module gets too big, many problems arise

- Name collisions
- It is good to organize modules in a bigger structure: *Packages*

Packages can be seen as "dotted module names"

- It is just a module that contains more modules
- Make life easier in big proyects
- The name `A.B` designates a submodule `B` in a package named `A`

Must contain an `__init__.py` file in the root directory

- Executed when the package is imported for the first time

Packages

Package concept (II)

Sound module structure

```
sound/                      Top-level package
      __init__.py           Initialize the sound package
      formats/              Subpackage for format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/              Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/              Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Packages

Importing a package (I)

Importing an individual module

```
import sound.effects.echo
sound.effects.echo.echofilter(input, output)
```

Alternative way to import an individual module

```
from sound.effects import echo
echo.echofilter(input, output)
```

Alternative way to import an individual module

```
from sound.effects.echo import echofilter
echofilter(input, output)
```

Packages

Importing a package (II)

Imagine we run `from sound import *`

- In theory, it would import the whole package
- In practice, it would take too much time

There is a convention to avoid waste of resources

- There may be a list `__all__` defined in `__init__.py`
- `__all__` contains the modules to be imported

sounds/effects/__init__.py

```
__all__ = ["echo", "surround", "reverse"]
```

Modules

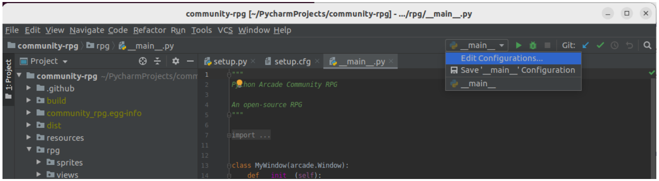
The `__main__.py` file

The file `__main__.py` is an alternative entry point in a package

- Useful in command line mode and IDEs

```
(In Linux console)
$ python3 -m mymodule
```

IDEs allow the configuration of multiple entry points


community-rpg [~/PycharmProjects/community-rpg] - .../rpg/__main__.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
community-rpg 〉 rpg 〉 __main__.py
__main__
Git:
Edit Configurations...
Save '__main__' Configuration
__main__
1: Project
Project
community-rpg ~/PycharmProjects/comm
.github
build
community_rpg.egg-info
dist
resources
rpg
sprites
views
setup.py
setup.cfg
__main__.py

```
"""
Python Arcade Community RPG

An open-source RPG
"""

import ...


class MyWindow(arcade.Window):
    def __init__(self):
```

Packages

Installing packages (I)

Command-line automatic tool: `pip` (sometimes `pip3`)

- Very similar to `apt-get` in Linux

pip usage (from OS terminal)

```
$ python -m pip install SomePackage
```

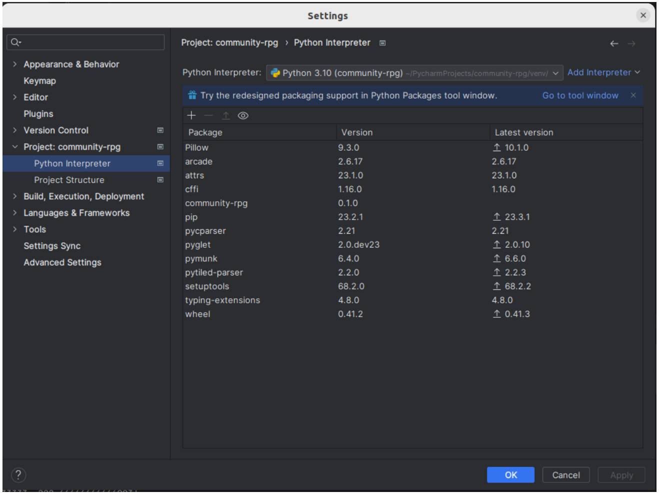
or

```
$ pip install SomePackage
```

```
$ pip install Pillow
```

Packages

Installing packages (II)



Python Interpreter: Python 3.10 (community-rpg) ~/PycharmProjects/community-rpg/venv/

Try the redesigned packaging support in Python Packages tool window.

Package	Version	Latest version
Pillow	9.3.0	↑ 10.1.0
arcade	2.6.17	2.6.17
attrs	23.1.0	23.1.0
cffi	1.16.0	1.16.0
community-rpg	0.1.0	
pip	23.2.1	↑ 23.3.1
pycparser	2.21	2.21
pyglet	2.0.dev23	↑ 2.0.10
pymunk	6.4.0	↑ 6.6.0
pytiled-parser	2.2.0	↑ 2.2.3
setuptools	68.2.0	↑ 68.2.2
typing-extensions	4.8.0	4.8.0
wheel	0.41.2	↑ 0.41.3

Packages

`requirements.txt` (I)

We need a tool to automatize our dependences: `requirements.txt`

- Plain text file with dependences
- Can use fixed or minimum version

`requirements.txt` can be used by different tools

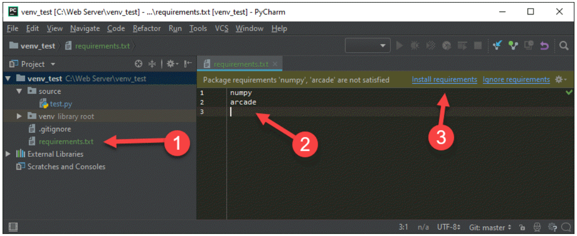
- `pip install -r requirements.txt`

requirements.txt

```
arcade==2.6.17
seaborn >=0.12
numpy
pandas
```

Packages

requirements.txt (II)


venv_test [C:\Web Server\venv_test] - ...\requirements.txt [venv_test] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
venv_test
requirements.txt
Project
venv_test C:\Web Server\venv_test
source
test.py
venv library root
.gitignore
requirements.txt
External Libraries
Scratches and Consoles
Install requirements
Ignore requirements
1
2
3
3:1 n/a UTF-8 Git: master

Package requirements 'numpy', 'arcade' are not satisfied

```
numpy
arcade
```

(Source)

Virtual environments

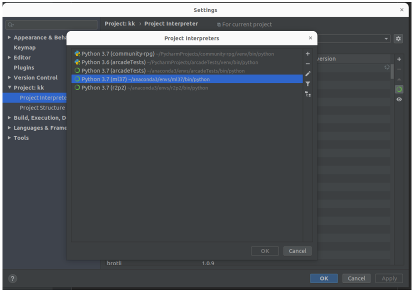
Versioning is problematic

- Python version (2.x, 3.x)
- Packages version

Solution: virtual environment

- Self-contained directory with a Python installation
- Particular version of Python and packages

Different solutions: venv and conda



```
Python 3.7 (community-rpg) ~/PycharmProjects/community-rpg/venv/bin/python
Python 3.6 (arcadeTests) ~/PycharmProjects/arcadeTests/venv/bin/python
Python 3.7 (arcadeTests) ~/anaconda3/envs/arcadeTests/bin/python
Python 3.7 (ml37) ~/anaconda3/envs/ml37/bin/python
Python 3.7 (r2p2) ~/anaconda3/envs/r2p2/bin/python
brotli 1.0.9
```

Great tool with `requirements.txt`!!!

Cool code examples

Example 1: Open a web browser

browser.py

```python
import webbrowser

url = input('Give me an URL: ')

webbrowser.open(url)
```

Cool code examples

Example 2: Create a thumbnail

thumbnail.py

```
from PIL import Image

size = (128, 128)
saved = "africa.jpg"

im = Image.open("africa.tif")
im.thumbnail(size)
im.save(saved)
im.show()
```

AFRICA
Map No. 4045 Rev. 7 UNITED NATIONS
November 2011
Department of Field Support
Cartographic Section

africa.jpg

(Source)

Cool code examples

Example 3: Send an email with Gmail

gmail.py

```python
"""The first step is to create an SMTP object,
each object is used for connection
with one server."""

import smtplib
server = smtplib.SMTP('smtp.gmail.com', 587)

# Next, log in to the server
server.login("youremailusername", "password")

# Send the mail
msg = "\nHello!" # /n separates the message from the headers
server.sendmail("you@gmail.com", "target@example.com", msg)
```

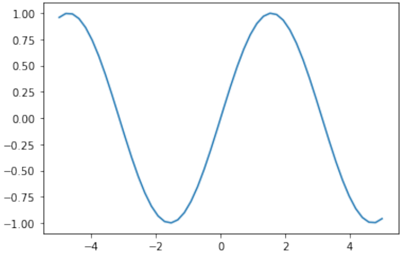
(Source)

Modules

Example 4: Plot

plot.py

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5)
plt.plot(x, np.sin(x))
```



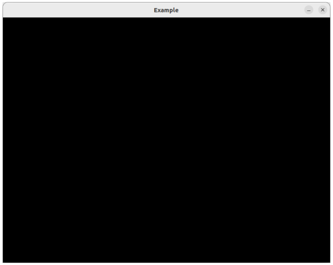
Modules

Example 5: Arcade

arcade.py

```
import arcade

WIDTH = 600
HEIGHT = 800

arcade.open_window(WIDTH, HEIGHT, "
    Example")

arcade.run()
```

- (API documentation)
- (Arcade source code)



Arcade
Introduction

Arcade is an easy-to-use 2D motor engine (i.e. a Python package)

- Created by Paul Vincent Craven
- Based on Python
- More or less painless game development
- Didactic
- Free software

Requires

- Python 3.6+
- OpenGL capable hardware

Dependences

- Pyglet - Mutimedia library for Python

(Arcade web site)

Arcade

Open a Window (I)

arcade.py

```python
import arcade

WIDTH = 600
HEIGHT = 800

arcade.open_window(WIDTH, HEIGHT, "Example")

arcade.run()
```

Arcade

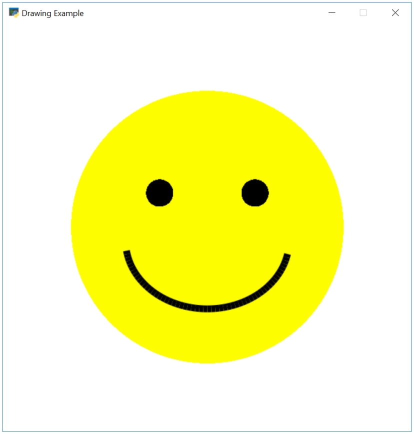
Drawing setup

drawing.py

```
import arcade

WIDTH = 600
HEIGHT = 800

arcade.open_window(WIDTH, HEIGHT,"Drawing Example")

arcade.set_background_color(arcade.color.WHITE)

arcade.start_render()

# Drawing here

arcade.finish_render()

arcade.run()
```

Arcade

Drawing (I)



(Source code)

Arcade

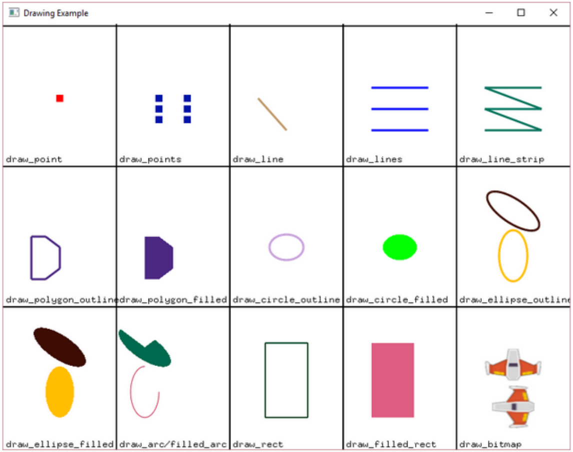
Drawing (II)

smile.py

```
# Draw the face
x = 300; y = 300; radius = 200
arcade.draw_circle_filled(x, y, radius, arcade.color.YELLOW)

# Draw the right eye
x = 370; y = 350; radius = 20
arcade.draw_circle_filled(x, y, radius, arcade.color.BLACK)

# Draw the left eye
x = 230; y = 350; radius = 20
arcade.draw_circle_filled(x, y, radius, arcade.color.BLACK)

# Draw the smile
x = 300; y = 280; width = 120; height = 100
start_angle = 190; end_angle = 350; line_width = 10
arcade.draw_arc_outline(x, y, width, height, arcade.color.BLACK,
                        start_angle, end_angle, line_width)
```

Arcade

Drawing primitives (I)



(Source code)

Arcade

Drawing primitives (II)

```
draw_rectangle_filled()
draw_rectangle_outline()        draw_arc_filled()
draw_lrtb_rectangle_filled()    draw_arc_outline()
draw_lrtb_rectangle_outline()
draw_xywh_rectangle_filled()    draw_circle_filled()
draw_xywh_rectangle_outline()   draw_circle_outline()

draw_polygon_filled()           draw_ellipse_filled()
draw_polygon_outline()          draw_ellipse_outline()

load_texture()                  draw_line()
draw_texture_rectangle()        draw_line_strip()
draw_xywh_rectangle_textured()  draw_lines()

draw_triangle_filled()          draw_point()
draw_triangle_outline()         draw_points()
```

Arcade

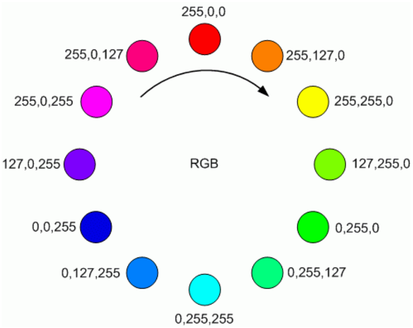
Colors

How can I know which colors has Arcade available?

- The reference API is your friend!
- (`arcade.color` reference documentation)

Arcade

Colors: RGB



(Source code)