Python packages

Read: Packages (/rltoken/Vn5hOrJ9lHds7we9udPnNg)

A Python file can be a **module** but when this file is in a folder, we call this folder a **package**.

File organization is really important in a big project. This means for Python: packages everywhere.

Compare with C

(file organization, not prototype vs code etc.)

In C: `#include "abs.h"`

In Python:

```
import abs
abs.my_abs(89)
```

or

```
from abs import my_abs
my_abs(89)
```

In C: `#include "my_math/abs.h"`

In Python:

```
from my_math.abs import my_abs
my_abs(89)
```

or

```
import my_math.abs
my_math.abs.my_abs(89)
```

Dotted module names == Path

Let's take this example of file organization:

```
my_script.py
my_math/
    abs.py
```

How can I use my function `my_abs(a)` from the file `abs.py` in `my_script.py` ?

- `import my_math/abs.py` => NO
- `import my_math/abs` => NO
- `import my_math.abs.py` => NO
- `import my_math.abs` => YES but you will use your function like that: `my_math.abs.my_abs(89)` => not friendly
- `from my_math.abs import my_abs` => YES YES YES! now you can use your function like that: `my_abs(89)`

Wait, does this really work?

NO! something is missing: the magic file `__init__.py`

Indeed, each folder **must** contain this file to be considered a package.

This file should be empty except if you want to import all the content of modules by using `*` .

More complicated?

```
my_script.py
my_math/
    __init__.py
    abs.py
    functions/
        __init__.py
        add.py
```

How can I use my function `my_add(a, b)` from the file `add.py` in `my_script.py` ?

```
from my_math.functions.add import my_add
```

Easy right?

`import *` is dangerous

Using `import *` is still considered bad practice in production code. In that case, `__init__.py` shouldn't be empty but must contain the list of modules to load:

```
my_script.py
my_math/
    __init__.py
    abs.py
    functions/
        __init__.py
        add.py
        sub.py
        mul.py
        div.py
```

```
$ cat my_script.py
from my_math.functions import *
print(add.my_add(1, 3))
print(mul.my_mul(4, 2))
print(div.my_div(10, 2))

$ cat my_math/__init__.py  # empty file
$ cat my_math/functions/__init__.py
__all__ = ["add", "mul"]

$ python3 my_script.py
3
8
Traceback (most recent call last):
  File "my_script.py", line 4, in <module>
    print(div.my_div(10, 2))
NameError: name 'div' is not defined
$
```

Relative versus Absolute import

In this example:

```
my_script.py
my_math/
    __init__.py
    abs.py
    positive.py
```

`positive.py` contains one function `def is_positive(n)` and this function uses `my_abs(n)`. How it's possible?

By importing: `from my_math.abs import my_abs` or `from abs import my_abs`

What the difference?

- `from abs import my_abs` is using a relative path between your file who imports and the module to import

- `from my_math.abs import my_abs` is using an absolute path between the file you execute and the (/) module to import

```
$ cat my_script.py
from my_math.positive import is_positive
print(is_positive(89))
print(is_positive(-89))
print(is_positive(333))

$ python3 my_script.py
True
False
True
$
```

Now, let's execute a file in `my_math` :

```
$ cd my_math ; cat test_positive.py
from positive import is_positive
print(is_positive(89))
print(is_positive(-89))
print(is_positive(333))

$ cat positive.py
from my_math.abs import my_abs

def is_positive(n):
    return my_abs(n) == n

$ python3 test_positive.py
Traceback (most recent call last):
  File "test_positive.py", line 1, in <module>
    from positive import is_positive
  File "/vagrant/my_math/positive.py", line 1, in <module>
    from my_math.abs import my_abs
ImportError: No module named 'my_math'
$
```

Ahh! If you are using an absolute path, you can't execute this module from another point as the "root" of your project.

Let's change to relative path:

```
$ cd my_math ; cat test_positive.py
from positive import is_positive
print(is_positive(89))
print(is_positive(-89))
print(is_positive(333))

$ cat positive.py
from abs import my_abs

def is_positive(n):
    return my_abs(n) == n

$ python3 test_positive.py
True
False
True
$
```