Clase03_1_modulos

March 27, 2024

0.0.1 Seminario de Lenguajes - Python

0.0.2 Módulos y paquetes en Python

1 Lo visto en clase

Un módulo es un archivo (con extensión .py) que contiene sentencias y definiciones.

Usualmente un programa puede estar formado por muchos módulos.

2 La sentencia import

Permite acceder a funciones y variables definidas en otro módulo.

[ ]:
```
import random
```

- Si el módulo contiene definiciones de funciones, sólo se importa estas definiciones, no las ejecuta.
- Para ejecutar una función definida en otro módulo **debo invocarla en forma explícita**.

[ ]:
```
random.randrange(10)
```

3 ¿Qué son los archivos .pyc?

- +Info en el sitio oficial, o podemos leer la PEP 3147
- **Afecta solo la velocidad de carga del archivo:** *"un programa no se ejecuta más rápido cuando es leído de un archivo .pyc que cuando es leído de un archivo .py"*

4 Espacios de nombres

- Recordamos que: > Un espacio de nombres **relaciona nombres con objetos**.
- En principio podemos pensar en tres espacios de nombres:
  - Locales
  - Globales
  - ___builtins___

5 Espacios de nombres

- Los espacios de nombres **se crean en diferentes momentos y tienen distintos tiempos de vida**.
- El espacio de nombres que contiene los nombres **___builtins___** se crea al iniciar el intérprete y nunca se elimina.
- El espacio de nombres **global** de un módulo se crea cuando se lee la definición de un módulo y normalmente también dura hasta que el intérprete finaliza.
- El espacio de nombres **local** a una función se crea cuando la función es llamada y se elimina cuando la función retorna.

6 Espacios de nombres y módulos

- Cada módulo tiene su propio espacio de nombres.
- ¿Cómo podemos saber los nombres definidos en un módulo?

[ ]:
```
for elem in dir(random):
    print(elem, end=" - ")
```

La función **dir** devuelve una lista ordenada de cadenas.

7 Exploremos espacios de nombres

- Probemos este código: ¿qué valor se imprime?

```
# modulo utils
var_x = 10
def print_var_x():
    print(var_x)

#modulo my_program
import utils

var_x = 20
utils.print_var_x()
```

[ ]:
```
import utils
```

[ ]:
```
dir(utils)
```

[ ]:
```
import builtins
dir(builtins)
```

8 Volvemos al import

- Cuando usamos la sentencia **import** se actualizan los espacios de nombres.

```
import mi_modulo
```

- En este caso, todos los recursos definidos dentro de **mi_modulo** serán locales a **mi_modulo**.
- Lo que se agrega al espacio de nombres es el nombre del módulo (**mi_modulo**).
- Para usarlo debo hacerlo con notación puntual.

```
mi_modulo.recurso
```

9 ¿Y acá?¿Qué nombre se agrega al espacio de nombres?

```
import mi_modulo as m

m.recurso
```

10 Otra forma de importar

```
from mi_modulo import una_funcion
```

- Sólo se importa **una_funcion** de **mi_modulo** (no el nombre del módulo).
- El único nombre que se agrega al espacio de nombres es **una_funcion**.

```
from mi_modulo import *
```

- En este caso, **todos los ítems** definidos en **mi_modulo** formarán parte del espacio de nombres actual.
- **Esta forma no está recomendada**: podrían existir conflictos de nombres.

11 Biblioteca de módulos estándar

- Existe un conjunto de módulos que vienen incluidos con el intérprete.
- Para usarlos se los debe importar en forma explícita (usando **sentencia import**).
- Ya usamos algunos, ¿cuáles?
- random, sys, string
- Hay muchos otros que nos ofrecen distinta funcionalidad.

[ ]:
```
import math

print(math.gcd(12,16))
print(math.sqrt(81))
print(math.pi)
```

[ ]:
```
import random

my_friends = ["Dibu Martinez", "Delpo", "Paula Pareto", "Luciana Aymar"]

print(random.choice(my_friends))
```

12 El módulo collections

- Define tipos de datos alternativos a los predefinidos dict, list, set, y tuple.
- **Counter**: es una **colección desordenada de pares claves-valor**, donde las claves son los elementos de la colección y los valores son la cantidad de ocurrencias de los mismos.

[ ]:
```
from collections import Counter

cnt = Counter(['red', 'blue', 'red', 'green', 'blue', 'blue'])
print(cnt)
```

[ ]:
```
letters = Counter('abracadabra')
print(letters)
print(letters.most_common(2))
```

13 Para probar luego

- El módulo **deque** ("double-ended queue"): permite implementar pilas y colas.

[ ]:
```
from collections import deque

d = deque('abcd')
# agrega al final
d.append("final")
# agrega al principio
d.appendleft("ppio")
# eliminar último
elem = d.pop()
# elimina primero
elem1=d.popleft()
d
```

14 El módulo sys

Entre otras cosas, define:

- **path:** las rutas donde buscar los módulos a cargar;
- **platform:** contiene información sobre la plataforma.

[ ]:
```
import sys
sys.platform
```

15 Veamos este otro ejemplo:

```
# modulo utils
def show_vowels(sentence):
    print(list(filter(lambda l: l.lower() in "aeiou", sentence)))
```

```
# modulo uso_utiles
import utils

utils.show_vowels("Holaaaa!!!!!!")
```

- Primero: ¿qué hace?

15.0.1 ¿Y si queremos invocar el módulo utiles (e invocar a la función vocales) desde la línea de comandos?¿Cómo les pasamos la cadena a analizar?

[ ]:
```
import sys
type(sys.argv)
```

- ¿De qué tipo es argv?
- ¿Qué valores contiene?
- sys.argv[0]?

Veamos un ejemplo en el IDE.

#

Los modulos pueden estar organizados en directorios.

16 Paquetes

Veamos el ejemplo de la documentación oficial de paquetes

```
import sound.effects.echo
from sound.effects import echo
```

16.0.1 ¿Qué contiene el archivo ___init___.py?

17 ¿Qué pasa si tenemos la siguiente sentencia?

```
from sound import *
```

- **___all___: es una variable que contiene una lista con los nombres de los módulos que deberían poder importarse cuando se encuentra la sentencia** from package import ***.

```
#Por ejemplo, en sound/effects/__init__.py

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

- Si **___all___** no está definida, **from sound.effects import *** no importa los submódulos dentro del paquete **sound.effects** al espacio de nombres.
- Un artículo para leer luego: Absolute vs Relative Imports in Python.