Clase 3_1_modulos

March 30, 2023

0.0.1 Seminario de Lenguajes - Python

0.0.2 Módulos en Python

#

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

#####

¿Algo nuevo?

1 Analicemos un ejemplo en el IDE

- ¿Cuántos archivos forman mi programa?
- ¿Cómo se relacionan?

2 La sentencia import

- Permite acceder a funciones y variables definidas en otro módulo.

[41]:
```
import random
```

- Sólo se importa las definiciones de las funciones. No las ejecuta.
- Para ejecutar una función **debo invocarla en forma explícita**.

[45]:
```
random.randrange(10)
```

[45]: 2

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?

[46]:
```
for elem in dir(random):
    print(elem, end=" - ")
```

```
BPF - LOG4 - NV_MAGICCONST - RECIP_BPF - Random - SG_MAGICCONST - SystemRandom -
TWOPI - _ONE - _Sequence - _Set - __all__ - __builtins__ - __cached__ - __doc__
- __file__ - __loader__ - __name__ - __package__ - __spec__ - _accumulate -
_acos - _bisect - _ceil - _cos - _e - _exp - _floor - _index - _inst - _isfinite
- _log - _os - _pi - _random - _repeat - _sha512 - _sin - _sqrt - _test -
_test_generator - _urandom - _warn - betavariate - choice - choices -
expovariate - gammavariate - gauss - getrandbits - getstate - lognormvariate -
normalvariate - paretovariate - randbytes - randint - random - randrange -
sample - seed - setstate - shuffle - triangular - uniform - vonmisesvariate -
weibullvariate -
```

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

7 Exploremos espacios de nombres

- Probemos este código: ¿qué valor se imprime?

```
#funciones
var_x = 10
def print_var_x():
    print(var_x)

#uso_funciones
import funciones
```

```
    var_x = 20
    funciones.print_var_x()
```

[47]:
```
import funciones
```

[48]:
```
dir(funciones)
```

[48]:
```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'dos',
 'tres',
 'uno']
```

[49]:
```
import builtins
dir(builtins)
```

[49]:
```
['ArithmeticError',
 'AssertionError',
 'AttributeError',
 'BaseException',
 'BaseExceptionGroup',
 'BlockingIOError',
 'BrokenPipeError',
 'BufferError',
 'BytesWarning',
 'ChildProcessError',
 'ConnectionAbortedError',
 'ConnectionError',
 'ConnectionRefusedError',
 'ConnectionResetError',
 'DeprecationWarning',
 'EOFError',
 'Ellipsis',
 'EncodingWarning',
 'EnvironmentError',
 'Exception',
 'ExceptionGroup',
 'False',
 'FileExistsError',
 'FileNotFoundError',
```

```
'FloatingPointError',
'FutureWarning',
'GeneratorExit',
'IOError',
'ImportError',
'ImportWarning',
'IndentationError',
'IndexError',
'InterruptedError',
'IsADirectoryError',
'KeyError',
'KeyboardInterrupt',
'LookupError',
'MemoryError',
'ModuleNotFoundError',
'NameError',
'None',
'NotADirectoryError',
'NotImplemented',
'NotImplementedError',
'OSError',
'OverflowError',
'PendingDeprecationWarning',
'PermissionError',
'ProcessLookupError',
'RecursionError',
'ReferenceError',
'ResourceWarning',
'RuntimeError',
'RuntimeWarning',
'StopAsyncIteration',
'StopIteration',
'SyntaxError',
'SyntaxWarning',
'SystemError',
'SystemExit',
'TabError',
'TimeoutError',
'True',
'TypeError',
'UnboundLocalError',
'UnicodeDecodeError',
'UnicodeEncodeError',
'UnicodeError',
'UnicodeTranslateError',
'UnicodeWarning',
'UserWarning',
```

```
'ValueError',
'Warning',
'ZeroDivisionError',
'__IPYTHON__',
'__build_class__',
'__debug__',
'__doc__',
'__import__',
'__loader__',
'__name__',
'__package__',
'__spec__',
'abs',
'aiter',
'all',
'anext',
'any',
'ascii',
'bin',
'bool',
'breakpoint',
'bytearray',
'bytes',
'callable',
'chr',
'classmethod',
'compile',
'complex',
'copyright',
'credits',
'delattr',
'dict',
'dir',
'display',
'divmod',
'enumerate',
'eval',
'exec',
'execfile',
'filter',
'float',
'format',
'frozenset',
'get_ipython',
'getattr',
'globals',
'hasattr',
```

```
'hash',
'help',
'hex',
'id',
'input',
'int',
'isinstance',
'issubclass',
'iter',
'len',
'license',
'list',
'locals',
'map',
'max',
'memoryview',
'min',
'next',
'object',
'oct',
'open',
'ord',
'pow',
'print',
'property',
'range',
'repr',
'reversed',
'round',
'runfile',
'set',
'setattr',
'slice',
'sorted',
'staticmethod',
'str',
'sum',
'super',
'tuple',
'type',
'vars',
'zip']
```

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 Importando módulos

```
import funciones

funciones.uno()
```

- La importación se realización **sólo una vez por sesión del intérprete**.
- Veamos sobre el ejemplo anterior.
- Si necesito volver a importar podemos usar **reload()** incluido en el **módulo importlib**.

```
import importlib
importlib.reload(funciones)
```

- +Info en la documentación oficial

11 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.

11.0.1 Veamos qué pasa cuando queremos importar un módulo que no existe:

[50]:
```
import pp
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[50], line 1
----> 1 import pp
```

```
ModuleNotFoundError: No module named 'pp'
```

11.0.2 ¿Dónde se busca los módulos?

- Directorio actual + otros directorios definidos en la variable de ambiente PYTHONPATH

[51]:
```
import sys
sys.path
```

[51]:
```
['/home/claudia/ownCloud/Materias/Python/2023/entorno3.11/teorias',
 '/usr/lib/python311.zip',
 '/usr/lib/python3.11',
 '/usr/lib/python3.11/lib-dynload',
 '',
 '/home/claudia/ownCloud/Materias/Python/2023/entorno3.11/lib/python3.11/site-
packages']
```

12 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.

[52]:
```
import math

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

```
4
9.0
3.141592653589793
```

[58]:
```
import random

amigos = ["Try", "Kalita","Kaze", "Khizha"]

print(random.choice(amigos))
```

```
Khizha
```

13 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.

[60]:
```
from collections import Counter

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

[63]:
```
cant_letras = Counter('abracadabra')
#cant_letras
print(cant_letras.most_common(2))
```

```
[('a', 5), ('b', 2)]
```

14 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
```

15 El módulo sys

- Entre otras cosas, define:
  - **exit([arg]):** sale del programa actual;
  - **path:** las rutas donde buscar los módulos a cargar;
  - **platform:** contiene información sobre la plataforma.

---

[64]:
```
import sys
sys.platform
```

[64]:
```
'linux'
```

16 Los nombres de los módulos

- Es posible acceder al nombre de un módulo a través de **___name___**

[65]:
```
print(__name__)
```

```
__main__
```

17 Tarea

- Averiguar cuándo un módulo se denomina **___main___**,

18 Paquetes

Veamos el ejemplo de la documentación oficial de paquetes

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

18.0.1 ¿Qué contiene el archivo ___init___.py?

19 ¿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.