Python (III)

Algoritmia
Grado en Ingeniería Informática
Universidad de Burgos

Juan José Rodríguez Diez

Contenido

1. Conveniencias adicionales
2. Ámbitos y espacios de nombres
3. Módulos

Expresiones condicionales

- *expr1* **if** *condicion* **else** *expr2*

```
param = n if n >= 0 else -n
result = foo(param)
```

```
result = foo(n if n >= 0 else -n)
```

```
print("aprobado" if nota >= 5 else "suspenso")
```

[Goodrich et al., 2013, pág. 42]

Comprensiones (I)

- Tarea habitual: producir una serie de valores a partir del procesamiento de otra serie.
- Comprensiones de listas:
  [*expresión* **for** *valor* **in** *iterable* **if** *condición*]
- Similar a:

```
1 resultado = []
2 for valor in iterable :
3     if condición :
4         resultado.append(expresión)
```

[Goodrich et al., 2013, pág. 43]

Comprensiones (II)

- Ejemplos:

```
1 cuadrados = [k * k for k in range(1, n + 1)]
2 divisores = [k for k in range(1, n + 1) if n % k == 0]
```

También hay comprensiones para conjuntos, generadores y diccionarios:

```
1 [k * k for k in range(1, n + 1)]       # lista
2 {k * k for k in range(1, n + 1)}       # conjunto
3 (k * k for k in range(1, n + 1))       # generador
4 {k : k * k for k in range(1, n + 1)} # diccionario
5
6 total = sum(k * k for k in range(1, n+1))
```

[Goodrich et al., 2013, pág. 43]

(Des)empaquetado de secuencias (I)

- Expresiones separadas por comas se consideran tuplas, aunque no estén encerradas en paréntesis.

```
>>> data = 2, 4, 6, 8
>>> print(data)
(2, 4, 6, 8)
```

- *Empaquetado automático* de una tupla.
- Devolver múltiples valores es una función:

```
return x, y
```

- En realidad devuelve una tupla.

[Goodrich et al., 2013, pág. 44]

(Des)empaquetado de secuencias (II)

- *Desempaquetado* automático.

```
1 a, b, c, d = range(7, 11)
2
3 cociente, resto = divmod(x, y)
```

- También puede aparecer en bucles **for**

```
1 for x, y in [ (7, 2), (5, 8), (6, 4) ]:
2         ....
3
4 for clave, valor in diccionario.items():
5         ....
```

[Goodrich et al., 2013, pág. 44]

Asignaciones simultáneas

- Combinación de empaquetado y desempaquetado.

```
1 x, y, z = 6, 2, 5
2
3 x, y = y, x
```

```
1 def fibonacci ():
2     a, b = 0, 1
3     while True:
4         yield a
5         a, b = b, a+b
```

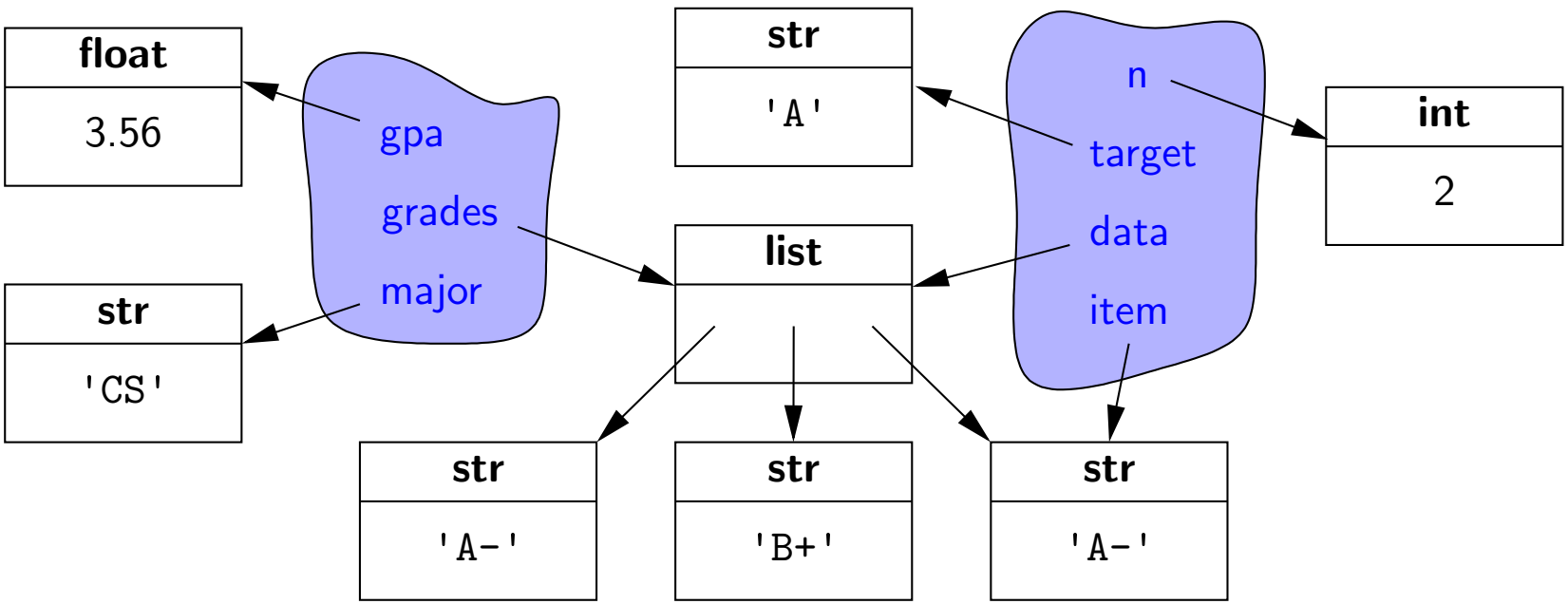
[Goodrich et al., 2013, pág. 45]

Ámbitos y espacios de nombres (I)

- Cuando se usa un nombre, debe estar asociado previamente a algún objeto. O se lanzará un *NameError*.
- *Resolución de nombres:* el proceso para determinar el valor asociado a un nombre.
- Cuando se asigna un valor a un identificador, se hace en un *ámbito* determinado.
- Ámbito *global*, asignaciones en el nivel superior.
- Asignaciones en funciones normalmente con un ámbito *local*.
- Un *espacio de nombres* representa un ámbito.
  - Gestiona los identificadores definidos en un ámbito determinado.

[Goodrich et al., 2013, pág. 46]

Ámbitos y espacios de nombres (II)



[Goodrich et al., 2013, pág. 46]

```
1 def count(data, target):
2     n = 0
3     for item in data:
4         if item == target: n += 1
5     return n
```

```
1 count(grades, 'A')
```

Ámbitos y espacios de nombres (III)

- La implementación de los espacios de nombres se realiza con los propios diccionarios de python.
- Para cada identificador, se asocia la cadena correspondiente con su valor.
- Función **dir**, muestra los nombres de los identificadores en un espacio de nombres dado.
- Función **vars** devuelve el diccionario completo.
- Sin argumentos toman el espacio de nombre más local en el que se ejecutan.
- Al buscar el valor asociado a un identificador, se empieza por el ámbito más local, se sigue por el siguiente ámbito externo.

[Goodrich et al., 2013, págs. 46-47]

Ámbitos y espacios de nombres (IV)

```
>>> x, y = 3, 7
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'x',
'y']
>>> vars()
{'__builtins__': <module 'builtins' (built-in)>,
'__package__': None, 'x': 3, 'y': 7, '__name__': '__main__',
'__doc__': None}

>>> def f(n): print(n, dir(), vars())

>>> f(9)
9 ['n'] {'n': 9}
```

Objetos de primera clase

- *First-class objects.*
- Instancias de un tipo que pueden ser asignadas a un identificador, pasado como parámetro o devueltos por una función.
- Los tipos presentados (e.g., **int**, **list**) son tipos de primera clase.
- Las funciones y clases también.

```
1 imprime = print
2 imprime("Hola")
```

- Las funciones pueden pasarse como argumento a otras funciones: **max**(*a*, *b*, *key*=**abs**)

[Goodrich et al., 2013, pág. 47]

Módulos (I)

- Varias funciones (e.g., **max**) y clases (e.g., **list**) definidas en el espacio de nombres predefinido.
- Bibliotecas adicionales, *módulos* que pueden ser *importados*.
- Algunas funciones matemáticas en el espacio de nombres predefinido: **abs**, **min**, **round**...
- Muchas más en el módulo *math*: *sin*, *sqrt*...
- Constantes: *pi*, *e*.
- Sentencia **import**.

[Goodrich et al., 2013, pág. 48]

Módulos (II)

- Importar elementos concretos de un módulo:

```
1 from math import pi, sqrt
```

- Importar todo de un módulo. No es recomendable, algunos nombres podrían estar ya en uso.

```
1 from math import *
```

- Importar el propio módulo.

```
1 import math
2 print(math.sqrt(math.pi))
```

[Goodrich et al., 2013, pág. 48]

Creación de módulos

- Fichero con la extensión `.py`.
- Las definiciones de ese fichero pueden ser importadas desde cualquier otro fichero `.py` del mismo directorio.
- Por ejemplo, si tenemos la función *contar* en el fichero `utilidades.py`:

```
1 from utilidades import contar
```

- Los comandos del nivel superior de un módulo se ejecutan la primera vez que se importa.
- Comandos que se ejecuten cuando el módulo se llama como script, pero no cuando se importa:

```
1 if __name__ == "__main__" :
```

[Goodrich et al., 2013, pág. 48]

Algunos módulos

array	Almacenamiento compacto de tipos primitivos.
collections	Estructuras adicionales y clases abstractas.
copy	Copias de objetos.
heapq	Colas de prioridad.
math	Constantes y funciones matemáticas.
os	Interacción con el sistema operativo.
random	Generación de valores aleatorios.
re	Expresiones regulares.
sys	Interacción adicional con el intérprete.
time	Medida de tiempo, retrasar un programa.

[Goodrich et al., 2013, pág. 49]

Valores pseudoaleatorios

seed(*hashable*)	Inicialización de la semilla con el valor hash
random()	Valor en $[0{,}0, 1{,}0)$
randint(*a*, *b*)	Valor en $[a, b]$
randrange(*start*, *stop*, *step*)	Entero en el rango indicado.
choice(*seq*)	Un elemento aleatorio de la secuencia
shuffle(*seq*)	Reordenación aleatoria

[Goodrich et al., 2013, pág. 50]

Referencias (I)

[Goodrich et al., 2013] El capítulo 1 contiene una introducción a Python.


Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2013).
Data Structures and Algorithms in Python.
Wiley.
http://bcs.wiley.com/he-bcs/Books?action=index&bcsId=8029&itemId=1118290275.