Lenguajes Dinámicos: Python

Tecnología de Programación

Adolfo Muñoz – Juan Magallón

Grado en Ingeniería Informática

Universidad Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

2023-01-20

Lenguajes Dinámicos

Lenguajes Dinámicos

Se suelen llamar lenguajes de programación dinámicos a un conjunto de lenguajes de alto nivel que implementan en tiempo de ejecución muchos comportamientos que otros lenguajes realizan en tiempo de compilación:

- la verificación de tipos
- la extensión del código del programa
- la definición de nuevos tipos de datos

Esas características pueden emularse en otros lenguajes suficientemente potentes, pero los lenguajes dinámicos dan soporte directo para ello.

Lenguajes Dinámicos

Es una diferenciación bastante ambigua, pero se suelen englobar en este tipo a muchos lenguajes conocidos:

- JavaScript (ECMAScript)
- Matlab
- Lua
- Ojective-C
- Perl
- PHP
- Ruby
- Python

Lenguajes Dinámicos

Suelen incorporar características comunes:

- alteración de la jerarquía de objetos en runtime
- clausuras: definiciones locales
- continuaciones
- reflectividad

Python

Python

Python es un lenguaje de programación de alto nivel, multiparadigma ya que soporta programación imperativa, orientación a objetos, y programación funcional.

Es un lenguaje interpretado, usa tipado dinámico, es fuertemente tipado y multiplataforma.

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI), en los Países Bajos.
El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python.

La versión 1.0 apareció en 1994, y ha evolucionado hasta la estandar actual 3.9, incorporando ideas de otros lenguajes como Haskell, Scheme o Icon.

Python es un lenguaje multiparadigma ya que soporta:

- programación imperativa.
- orientación a objetos.
- programación funcional.

Utiliza gestión automática de memoria mediante una cuenta de referencias.

Una característica importante es la asociación dinámica de símbolos, tanto para variables como funciones o métodos.

Se puede extender fácilmente mediante módulos en C o C++, y se puede utilizar también de forma fácil desde esos lenguajes, por ejemplo para embeber un intérprete en un programa más grande.

Python es un lenguaje interpretado.

El intérprete se llama 'python' en la mayoria de las plataformas, y funciona en modo interactivo:

```
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
...
Be careful not to fall off!
```

Hello World

El Hello World en Python es muy simple:

hello.py

```
1  print("Hello, world!!")
```

Podemos ejecutarlo con:

```
1  python hello.py
```

O bien convertirlo en un script ejecutable:

hello.py

```
1  #!/usr/bin/python
2  print("Hello, world!!")
```

(no olvides el `chmod +x hello.py`...)

En python los comentarios se marcan con el símbolo '#'.

En python es importante el sangrado, en muchas construcciones es la única forma de marcar los límites de los bloques, no existen delimitadores específicos:

```
1  # funcion factorial
2  def factorial(x):
3      if x == 0:
4          return 1
5      else:
6          return x * factorial(x - 1)
```

Las variables son dinámicas, lo que significa que no se declaran y que pueden apuntar a distintos tipos de valores a lo largo de su vida:

```
>>> a = 7
>>> print(a)
7
>>> a = "Hola"
>>> print(a)
Hola
```

Cuando una variable deja de apuntar a un valor, el sistema de cuenta de referencias determinará si se elimina de memoria, aunque podemos borrar manualmente una variable con `del`.

Tipos de datos simples:

Python 3.x

bool	Booleano		True o False
int	Entero	Precisión arbitraria	4569667843
float	Decimal	Doble precisión	3.1415927
complex	Complejo	Doble precisión	1+2j
str	Cadena	Unicode	'Cadena'
bytes	Datos binarios		

Tipos de datos compuestos (heterogéneos):

list	Mutable	[4.0, 'Cadena', True]
tuple	Inmutable	(4.0, 'Cadena', True)
set	Mutable	{4.0, 'Cadena', True} , set(çacatua")
frozenset	Inmutable	frozenset([4.0, 'Cadena', True]) , frozenset(çacatua")
dict	Mutable	{'key1': 1.0, 'key2': False}

Listas y tuplas se diferencian por la mutabilidad.
Se puede acceder a sus elementos de distintas formas:

```
1  l = [1,2.0,"abc",4]
```

- indexación directa: de 0 a n-1

```
1  >>> l[1]
2  2.0
```

- indexación inversa: de -1 a -n

```
1  >>> l[-1]
2  4
```

- *slicing*

```
1  >>> l[1:3]
2  [2.0, 'abc']
3  >>> l[-3:-1]
4  [2.0, 'abc']
```

En Python todo son objetos, en particular las listas:

```
l = [1,2.0,"abc",4]
```

```
>>> l.append(9)
>>> l
[1, 2.0, 'abc', 4, 9]
>>> l.extend(['C','D'])
>>> l
[1, 2.0, 'abc', 4, 9, 'C', 'D']
>>> l.insert(2,'I')
>>> l
[1, 2.0, 'I', 'abc', 4, 9, 'C', 'D']
```

En Python todo son objetos, en particular las listas:

```
1  l = [1,2.0,"abc",4]
```

- `len(l)`: longitud
- `l.remove(x)`, `del l[i]`: eliminar elementos
- `l.pop()`: quita el último elemento
- `l.index(x)`, `l.count(x)`: búsquedas
- `l.sort()`, `l.reverse()`: ordenación

Python incorpora la idea de las listas por comprensión con el modelo de Haskell:

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

squares = [x**2 for x in range(10)]
```

Pueden contener bucles anidados y condiciones:

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Pueden usar variables locales definidas previamente:

```
>>> # flatten a list using a listcomp with two 'for'
>>> mat = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for row in mat for num in row]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

La definición de funciones es simple:

```
1  def suma(x, y = 2):
2          return x + y
```

Pueden tener valores por defecto para los argumentos y utilizar la asociación dinámica, es decir, depender de variables o funciones que aún no existen.
Existe la notación lambda para definir funciones:

```
1  suma = lambda x, y = 2 : x + y
```

Condicional: if

```
1  lenguaje = "Python"
2  if lenguaje == "C":
3       print "Lenguaje C"
4  elif lenguaje == "Python":
5       print "Lenguaje Python"
6  else:
7       print "Lenguaje indefinido"
```

Iteración: for

El bucle for recorre un objeto iterable: listas, tuplas o generadores

```
1  lista = ["a", "c", "b"]
2  for i in lista:
3       print i
4  for i in range(5):
5       print i
6  for i in reversed(range(1,10,2)):
7       print i
8  for i in sorted(lista):
9       print i
```

Iteración: while

La iteración indefinida es similar a otros lenguajes:

```
1  numero = 0
2  while numero < 10:
3       numero += 1
4       print(numero)
```

Iteración: else

Los bucles pueden tener una sección `else`: se ejecuta cuando el bucle termina normalmente pero no después de `break`

```
for n in range(2, 10):
  for x in range(2, n):
    if n % x == 0:
      print(n, 'equals', x, '*', n//x)
      break
  else:
    print(n, 'is a prime number')
```

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Las clases se definen con la palabra clave **class**, seguida del nombre de la clase y, si hereda de otra clase, el nombre de esta.

El constructor se denomina `__init__`.

El objeto se declara como parámetro de forma explicita mediante `self`.

Los atributos no se declaran, se crean automáticamente.

```
1  class Complex:
2      r = 0.0
3      i = 0.0
4      def __init__(self, realpart, imagpart):
5          r = realpart
6          i = imagpart
7
8  a = Complex(1,2)
9  a.r, a.i
10 (0.0, 0.0)
```

```
1 class Pair:
2     def __init__(self, f, s):
3         self.f = f
4         self.s = s
```

Ejemplos

```python
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

```
1  def fib(n):
2  """Return a list containing the Fibonacci series up to n."""
3    result = []
4    a, b = 0, 1
5    while a < n:
6      result.append(a)
7      a, b = b, a+b
8    return result
9
10 fib(100)
11 [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
12 print(fib.__doc__)
```

Lenguajes Dinámicos: Python

Tecnología de Programación

Adolfo Muñoz – Juan Magallón

Grado en Ingeniería Informática

Universidad Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

2023-01-20