MOOC de Introducción a sage. Funciones y Objetos

Leandro Marín

Definición

- Una función tiene la siguiente estructura:

```
def nombre_de_funcion(a,b,c):
  operaciones
  operaciones
  return x
```

Tal y como sucedía en los `while`, `for` e `if`, para iniciar el bloque de código que define la función hay que poner `:` e indentar el bloque completo.

Definición

- Una función tiene la siguiente estructura:

```
def nombre_de_funcion(a,b,c):
  operaciones
  operaciones
  return x
```

  Tal y como sucedía en los `while`, `for` e `if`, para iniciar el bloque de código que define la función hay que poner `:` e indentar el bloque completo.

- La función puede tomar 0 parámetros y si no devuelve nada, lo que hace realmente es devolver automáticamente el valor especial `None`.

Ejemplo I

```
def f(n):
  if n == 0:
    return 1
  else:
    return n*f(n-1)

print f(20)
```

Nos dará 2432902008176640000.

- Esta función toma un parámetro `n` y dependiendo de su valor nos devuelve 1 o hace una llamada recursiva para `n-1` y calcula el resultado.

Ejemplo II

```
def rota(L):
  return L[1:]+[L[0]]

print rota([1,2,3])
```

Nos dará `[2, 3, 1]`.

- Esta función toma como marámetro una lista `L` y nos devuelve la lista que extrae el primer elemento de `L` y lo pone en la posición final.

Definición

- Las clases nos permiten crear nuestros propios tipos de datos estructurados así como las funciones que operan sobre ellos (a las que llamaremos métodos).

Definición

- Las clases nos permiten crear nuestros propios tipos de datos estructurados así como las funciones que operan sobre ellos (a las que llamaremos métodos).
- La definición de una clase se hará poniendo

```
class NombreDeClase:
  linea 1
  linea 2
  ...
  linea n
```

en las distintas lineas iremos definiendo los distintos metodos que constituyen la clase.

Ejemplo : Puntos en el Plano

- Vamos a construir una clase sencilla para representar puntos en el plano.

Ejemplo : Puntos en el Plano

- Vamos a construir una clase sencilla para representar puntos en el plano.
- Los puntos estarán dados por dos coordenadas (x, y) y sobre ellos realizaremos una serie de operaciones.

Ejemplo : Puntos en el Plano

- Vamos a construir una clase sencilla para representar puntos en el plano.
- Los puntos estarán dados por dos coordenadas (x, y) y sobre ellos realizaremos una serie de operaciones.
- Lo primero que haremos será crear un constructor de la clase, es decir, una función tal que cuando escribamos `P = Punto(3,-1)` nos genere un objeto punto `P` con esas coordenadas.

Ejemplo : Puntos en el Plano

- Vamos a construir una clase sencilla para representar puntos en el plano.
- Los puntos estarán dados por dos coordenadas (x, y) y sobre ellos realizaremos una serie de operaciones.
- Lo primero que haremos será crear un constructor de la clase, es decir, una función tal que cuando escribamos `P = Punto(3,-1)` nos genere un objeto punto `P` con esas coordenadas.
- Eso se hace con el método `__init__`.

El Constructor de la Clase

- El constructor es un método especial que se llama `__init__` y al que se le deben pasar como parámetros uno especial, llamado `self` y luego los necesarios para construir el objeto, en este caso las coordenadas del mismo.

```
class Punto:
  def __init__(self,x,y):
    self.x = x
    self.y = y
P = Punto(2,-3)
print P.x
```

El Constructor de la Clase

- El constructor es un método especial que se llama `__init__` y al que se le deben pasar como parámetros uno especial, llamado `self` y luego los necesarios para construir el objeto, en este caso las coordenadas del mismo.

```
class Punto:
  def __init__(self,x,y):
    self.x = x
    self.y = y
P = Punto(2,-3)
print P.x
```

- Para inicializar los datos correspondientes al objeto punto debemos hacerlo con `self`, el operador `.` y el nombre que queramos dar a los datos.

El Constructor de la Clase

- El constructor es un método especial que se llama `__init__` y al que se le deben pasar como parámetros uno especial, llamado `self` y luego los necesarios para construir el objeto, en este caso las coordenadas del mismo.

```
class Punto:
  def __init__(self,x,y):
    self.x = x
    self.y = y
P = Punto(2,-3)
print P.x
```

- Para inicializar los datos correspondientes al objeto punto debemos hacerlo con `self`, el operador `.` y el nombre que queramos dar a los datos.
- Para crear un objeto se llamará al nombre de la clase con los parámetros elegidos.

Definiendo Métodos

- Antes de seguir viendo las características de este método especial `__init__` vamos a ver cómo se definen otros métodos y luego volveremos a él.

Definiendo Métodos

- Antes de seguir viendo las características de este método especial `__init__` vamos a ver cómo se definen otros métodos y luego volveremos a él.
- Vamos a crear un método llamado `norte` que nos mueva el punto una unidad hacia arriba del plano.

Definiendo Métodos

- Antes de seguir viendo las características de este método especial `__init__` vamos a ver cómo se definen otros métodos y luego volveremos a él.
- Vamos a crear un método llamado `norte` que nos mueva el punto una unidad hacia arriba del plano.
- Aunque hemos dicho que `norte` no tendría parámetros, por ser método de clase debe tener el parámetro `self` que nos permite acceder a la información del objeto.

```
class Punto:
  def __init__(self,x,y):
    self.x = x
    self.y = y
  def norte(self):
    self.y = self.y + 1

P = Punto(2,-3)
print "Estamos en ({0},{1})".format(P.x,P.y)
P.norte()
print "Ahora estamos en ({0},{1})".format(P.x,P.y)
```

Nos dará

```
Estamos en (2,-3)
Ahora estamos en (2,-2)
```

Añadiendo Parámetros

- Supongamos que en lugar de movernos una unidad al norte, queremos movernos `n` unidades.

Añadiendo Parámetros

- Supongamos que en lugar de movernos una unidad al norte, queremos movernos `n` unidades.
- Para eso debemos poner `n` tras el parámetro *invisible* `self`, la función tendrá dos parámetros pero accederemso a ella poniendo `P.norte(7)`.

```
class Punto:
  def __init__(self,x,y):
    self.x = x
    self.y = y
  def norte(self,n):
    self.y = self.y + n

P = Punto(2,-3)
print "Estamos en ({0},{1})".format(P.x,P.y)
P.norte(7)
print "Ahora estamos en ({0},{1})".format(P.x,P.y)
```

Nos dará

```
Estamos en (2,-3)
Ahora estamos en (2,4)
```

Otros métodos especiales

- Podemos definir tantos métodos como queramos, pero hay algunos que tienen algunas características especiales tal y como sucedía con el constructor `__init__`.

Otros métodos especiales

- Podemos definir tantos métodos como queramos, pero hay algunos que tienen algunas características especiales tal y como sucedía con el constructor `__init__`.
- Por ejemplo, al escribir en la pantalla, estamos incluyendo un código que repetimos varias veces:

```
print "... ({0},{1})".format(P.x,P.y)
```

Otros métodos especiales

- Podemos definir tantos métodos como queramos, pero hay algunos que tienen algunas características especiales tal y como sucedía con el constructor `__init__`.
- Por ejemplo, al escribir en la pantalla, estamos incluyendo un código que repetimos varias veces:

```
print "... ({0},{1})".format(P.x,P.y)
```

- Podemos indicar que esa es la representación de los puntos en una cadena de caracteres definiendo el método `__str__`.

```
class Punto:
  def __init__(self,x,y):
    self.x = x
    self.y = y
  def __str__(self):
    return "({0},{1})".format(self.x,self.y)
  def norte(self,n):
    self.y = self.y + n

P = Punto(2,-3)
print P
P.norte(7)
print "Ahora estamos en {0}".format(P)
```

Nos dará

```
(2,-3)
Ahora estamos en (2,4)
```

Sobrecarga de Operadores

- Utiliando métodos especiales podemos definir operaciones sobre nuestros objetos.

Sobrecarga de Operadores

- Utiliando métodos especiales podemos definir operaciones sobre nuestros objetos.
- Por ejemplo, si queremos definir la suma de un punto con un vector representado por una lista de dos coordenadas podemos hacerlo definiendo la operación `__add__`.

Sobrecarga de Operadores

- Utiliando métodos especiales podemos definir operaciones sobre nuestros objetos.
- Por ejemplo, si queremos definir la suma de un punto con un vector representado por una lista de dos coordenadas podemos hacerlo definiendo la operación `__add__`.
- En este caso la operación nos devuelve un dato, que es el nuevo punto.

```
class Punto:
  def __init__(self,x,y):
    self.x = x
    self.y = y
  def __str__(self):
    return "({0},{1})".format(self.x,self.y)
  def __add__(self,v):
    self.x = self.x + v[0]
    self.y = self.y + v[1]
    return self

P = Punto(-1,4)
Q = P+[2,2]
print Q
```

Nos dará

```
(1,6)
```

Conclusión

- Hemos visto muy, muy, muy por encima algunas de las características de las clases y objetos.

Conclusión

- Hemos visto muy, muy, muy por encima algunas de las características de las clases y objetos.
- Este es un tema del que podríamos hablar mucho, temas como elementos estáticos de las cases o herencia, pero en este curso tan básico no nos podemos adentrar ahí.

Conclusión

- Hemos visto muy, muy, muy por encima algunas de las características de las clases y objetos.
- Este es un tema del que podríamos hablar mucho, temas como elementos estáticos de las cases o herencia, pero en este curso tan básico no nos podemos adentrar ahí.
- La orientación a objetos es una forma muy adecuada para crear código matemático, todo lo que manejamos son objetos y de una u otra forma estamos haciendo uso de ella aunque no seamos conscientes.