Programación para Física y Astronomía

Departamento de Física.

June 7, 2022

Universidad Andrés Bello®

python powered

Resumen

Funciones en Python

Clases y Objetos

- Definir una clase

Diseño orientado a objetos

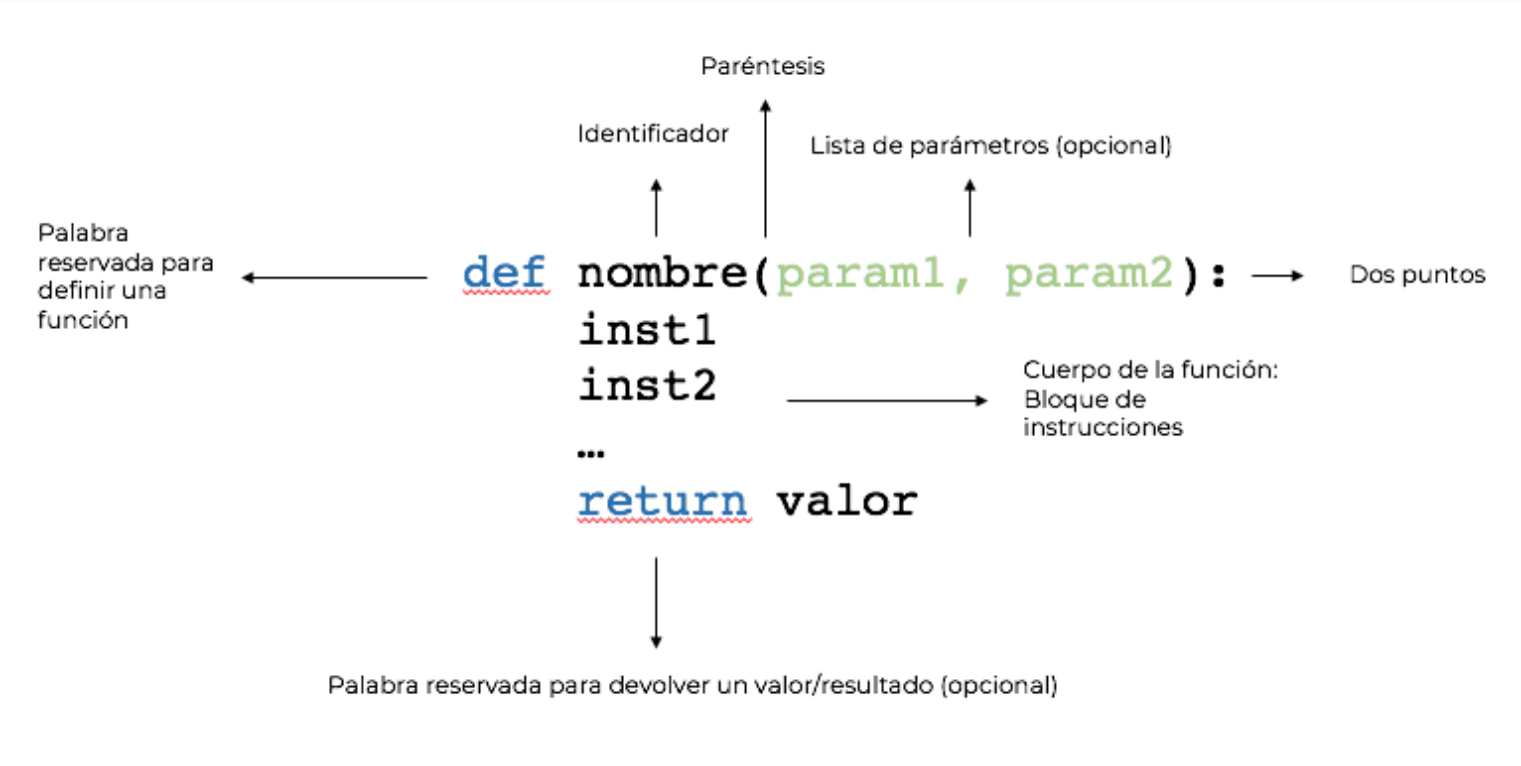
- Polimorfismo en Python

Funciones en Python

Funciones

- Las **funciones** son estructuras esenciales de un código.
- Una función es un grupo de instrucciones que constituyen una unidad lógica del programa, y resuelven un problema concreto.
- Hemos utilizado funciones ya, por ejemplo la función random de la libreria *`random`*, la función *`sin`*, de la librería math, y muchas más de la librería *`numpy`*.
- Una función siempre es de gran ayuda ya que nos permite organizar nuestros códigos de forma más sencilla.

Como se construye una función



Llamando funciones

Para llamar una función, ya definida, esta puede o no requerir argumentos, los que se entregan entre paréntesis redondos, luego del nombre de la función. Consideremos el siguiente ejemplo:

```
1  #Definimos la función. Notar la indentación
2  def multiplica_por_cinco(numero):
3      print(f'{numero} * 5 = {numero * 5}')
4
5  #Comienza nuestro programa
6  print('Comienzo del programa')
7  multiplica_por_cinco(7)   #entregamos el argumento de la función
8  print('Siguiente')
9  multiplica_por_cinco(113)
10 print('Fin')
```

Retornando valores

Una función puede retornar uno o más valores utilzando la instrucción return al final de la implementación:

```
1  #Definimos la función. Notar la indentación
2  def eleva_a3(numero):
3      return numero*numero*numero
4
5  #Comienza nuestro programa
6  n = 5
7  r = eleva_a3(n) #Acá la función retorna el valor y lo asigna a r.
8
9  print(f'el valor de {n} elevado a 3 es {r}')
```

Retornando valores

Retornando más valores:

```
#Definimos la función. Notar la indentación
def potencias_hasta_diez(numero):
    l = []
    for i in range(10):
        l.append(pow(numero,i))
    return l

#Comienza nuestro programa
n = 5
r = potencias_hasta_diez(n) #Acá la funcion retorna una lista

print(f'La lista de potencias de {n} es {r}')
```

Alcance o ámbito

Parámetros definidos dentro de una función sólo existen dentro, no pueden ser llamados fuera del ámbito de la función.

```
1  #Definimos la función. Notar la indentación
2  def elevar_a_diez(numero):
3      h = numero*numero*numero*numero*numero
4      return h
5
6  #Comienza nuestro programa
7  n = 5
8  print(h) #<- Esta linea mostrará un error
9  r = elevar_a_diez(n) #Acá la funcion retorna una lista
10 print(f'{n} elevado a 10 es {r}')
```

Despues de ver el error cambie el nombre de la variable *n* a *h*.

Clases y Objetos

Objetos

- Definiremos un objeto como un tipo de datos *active* que sabe algo y que puede hacer algo.
- De forma más precisa, un objeto consiste en:
    - Un set de información relacionada.
    - Un set de operaciones para manipular dicha información.
- La información es almacenada dentro del objeto en *variables de instancia*.
- Las operaciones (llamadas métodos), son funciones que viven dentro del objeto.
- De manera colectiva, las variables de instancia y los métodos son llamados atributos del objeto.

Clases y objetos

- **Clase:** Esquema abstracto de un tipo de objeto. Por ejemplo, un automóvil.
- **Objeto:** Un individuo o ejemplar de una clase. Por ejemplo, un Fiat 600, rojo.
- **Atributo:** Una propiedad de un objeto. Por ejemplo, el número de puertas, color, tipo de motor.
- **Método:** Una acción que un objeto puede realizar. Por ejemplo acelerar, frenar, poner reversa, etc.

Clases y objetos

- **Herencia:** Una clase puede heredar métodos y atributos de una clase más general. Por ejemplo la clase automóvil hereda algunas de sus características de la clase vehículo (un vehículo puede ser un avión, un barco, etc).
- **Interfaz:** La apariencia externa de un objeto, definida por sus atributos y métodos públicos. El conjunto de interfaces en una librería se denomina API (*Application Programming Interface*)
- **Implementación:** La estructura interna del objeto, definida por sus atributos y métodos privados.

Definición de una clase en Python 3

En Python, un método se diferencia de una función, en que lleva un argumento extra, por convención llamado ***self***.

```
1  def dist(a,b):
2      return sqrt((a[0]-b[0])**2 + (a[1]-b[1])**2)
3  class Circulo:
4      centro, r = (0.0, 0.0), 1.0
5      def Diametro(self): return 2.0*self.r
6      def Area(self): return pi*self.r**2
7      def Perimetro(self): return 2.0*pi*self.r
8      def EstaDentro(self, a):
9          return (dist(a, self.centro) < self.r)
10 c1 = Circulo()
11 c1.r = 5.0
12 c1.centro = (0.3, 0.5)
13 print(c1.Area())
14 print(c1.Perimetro())
15 print(c1.EstaDentro((3.0, 3.0)))
```

Constructores

- Para no tener que crear un objeto, y asignar sus atributos de forma separada, se usan métodos especiales llamados constructores, que reciben argumentos.
- En Python el constructor siempre se llama *__init__*

```
1  class Circulo:
2      centro, r = (0.0, 0.0), 1.0
3      def __init__(self, r0, c0):
4          self.r, self.centro = r0,c0
5  # ...
```

Diseño orientado a objetos

Diseño

- Orientación a objetos es una forma clara de poner orden en la complejidad de un programa.
- Naturalmente hace un programa modular, cada objeto es una unidad de código independiente.
- Incentiva la encapsulación, es decir, una parte del código no necesita saber cómo funciona otra parte.
- Permite la técnica del polimorfismo, es decir, programar algoritmos genéricos reutilizables en distintas situaciones.

Herencia

```
1  class Mamifero:
2      tieneCola, color = False, 'gris'
3      def Gritar(self):
4          pass
5  class Gato(Mamifero): #Heredemos de Mamifero
6      tieneCola, tieneCascabel = True, False
7      nombre, color = 'Garfield', 'naranja'
8      def Gritar(self):
9          print(self.nombre, ' dice Miau!')
10 class Perro(Mamifero):
11     tieneCola, nombre = True, 'Bobby'
12     def Gritar(self):
13         print(self.nombre, ' dice Guau!')
```

Polimorfismo

```
#g es ejemplar de clase Gato
g = Gato()
g.nombre = 'jim'
g.color = 'pardo'
g.tieneCascabel = True
#p es ejemplar de la clase Perro
p = Perro()
p.nombre = 'dexter'
p.color = 'cafe'
#Tanto p como g entienden el metodo Gritar
#pero cada uno lo implementa distinto
for x in [g, p]: x.Gritar()
```

Polimorfismo en Python

- En *C++* deben declararse en la clase base con la palabra virtual todos los métodos que se espera sean redefinidos en las clases especializadas.
- Python, en ese sentido, es más como Java, todos los métodos pueden ser redefinidos (virtuales por omisión).

Polimorfismo en Python

- Existen también métodos completamente abstractos, que no tienen sentido definirlos (ni siquiera vacíos) en una clase base. *Cómo sería el método CalcularPerimetro() de la clase FiguraGeometrica?*
- Uno está obligado a definir estos métodos en las clases especializadas. A éstos en *C++* y Java se les llama métodos virtuales puros.
  Python no tiene métodos virtuales puros. La manera pitónica de conseguir polimorfismo es un ingenioso concepto llamado ***duck typing***.

Duck Typing

Duck Typing

→ *Si camina como un pato y hace quack como pato, yo lo llamaría un pato.*

- No impora si una clase deriva o no de otra, lo realmente importante es cómo se ve por fuera (*interfaz*). Una clase debería poder hacerse pasar por otra si imita los mismos atributos y métodos.
- No es necesaria la herencia para conseguir polimorfismo, lo que permite una flexibilidad equivalente a los template de *C++*, pero sin la complicación de una nueva sintaxis.

Duck Typing

```
1  class Duck:
2      def quack(self):
3          print("Quaaaack!")
4      def feathers(self):
5          print("The duck has white and gray feathers.")
6  class Person:
7      def quack(self):
8           print("The person imitates a duck.")
9      def feathers(self):
10         print("The person takes a feather from
11         \\ the ground and shows it.")
12     def name(self):
13         print("John smith")
14 def in_the_forest(duck):
15     duck.quack()
16     duck.feathers()
```

Duck Typing

```
def game():
    donald = Duck()
    john = Person()
    in_the_forest(donald)
    in_the_forest(john)
game()
```

Fin