Capacitarte

Material Imprimible

Curso de Python

Módulo 5

El parámetro self

El parámetro self es una referencia a la instancia actual de la clase y se utiliza para tener acceso a las variables(propiedades) que pertenecen a la clase.

No tiene que ser nombrado self, se puede llamar como quieras, pero tiene que ser el primer parámetro de cualquier función de la clase:

Utilice las *palabras mysillyobject* y *abc* en lugar de *self*:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modificar propiedades de objeto

Puede modificar propiedades en objetos como este:

Establezca la edad de p1 en 40:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
```

```
  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.age = 40

print(p1.age)
```

Eliminar propiedades de objeto

Puede eliminar propiedades en objetos mediante la palabra clave: del

Elimine la propiedad age del objeto p1: Nos dará un error

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1.age

print(p1.age)
```

Eliminar objetos

Puede eliminar objetos utilizando la palabra clave:del

Elimine el objeto p1: Nos dará error

```
class Person:
```

```
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1

print(p1)
```

La Declaración de pass

class las definiciones no pueden estar vacías, pero si por alguna razón tiene una definición sin contenido, coloque la instrucción para evitar recibir un error.classpass

```
class Person:
  pass
```

Herencia de Python

La herencia nos permite definir una clase que hereda todos los métodos y propiedades de otra clase.

Clase primaria es la clase de la que se hereda, también denominada clase base.

Clase secundaria es la clase que hereda de otra clase, también denominada clase derivada.

Crear una clase principal

Cualquier clase puede ser una clase primaria, por lo que la sintaxis es la misma que crear cualquier otra clase:

Cree una clase denominada Person, with y properties, y un método: firstname lastname printname

```
class Person:
```

```
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)
#Utilice la clase Person para crear un objeto y luego ejecute el método printname:
x = Person("John", "Doe")
x.printname()
```

Crear una clase secundaria

Para crear una clase que herede la funcionalidad de otra clase, envíe la clase primaria como parámetro al crear la clase secundaria:

Cree una clase denominada Student, que heredará las propiedades y métodos de la clase:

```
class Student(Person):
  pass
```

Nota: Utilice la palabra clave cuando no desee agregar ninguna otra propiedad o método a la clase.pass

Ahora la clase Student tiene las mismas propiedades y métodos que la clase Person.

Utilice la clase para crear un objeto y, a continuación, ejecute el método:Studentprintname

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  pass
x = Student("Mike", "Olsen")
```

x.printname()

Añadir la función __init__()

Hasta ahora hemos creado una clase secundaria que hereda las propiedades y métodos de su elemento primario.

Queremos agregar la función __init__() a la clase secundaria (en lugar de la palabra clave). pass

Nota: La función __init__() se llama automáticamente cada vez que se utiliza la clase para crear un nuevo objeto. __init__()

Agregue la función __init__() a la clase: Student

```
class Student(Person):
  def __init__(self, fname, lname):
    #agregar propiedades etc.
```

Al agregar la función__init__(), la clase secundaria ya no heredará la función __init__() del elemento primario.

Nota: La función__init__() del elemento secundario **reemplaza** la herencia de la función __init__()del elemento primario.

Para conservar la herencia de la función__init__() del elemento primario, agregue una llamada a la función del elemento primario: __init__()

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
          print(self.firstname, self.lastname)

class Student(Person):
```

```
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

Ahora hemos agregado correctamente la función __init__() y hemos mantenido la herencia de la clase primaria, y estamos listos para agregar funcionalidad en la función.__init__()

Utilice la función super()

Python también tiene una función super() que hará que la clase secundaria herede todos los métodos y propiedades de su elemento primario:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

Mediante el uso de la función super(), no es necesario utilizar el nombre del elemento primario, heredará automáticamente los métodos y propiedades de su elemento primario.

Añadir propiedades

Agregue una propiedad llamada graduationyear a la clase: Student

```
class Person:
```

```
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019

x = Student("Mike", "Olsen")
print(x.graduationyear)
```

En el ejemplo siguiente, el año debe ser una variable y pasar a la clase al crear objetos de alumno. Para ello, añada otro parámetro en la función __init__():2019Student
Agregue un parámetro y pase el año correcto al crear objetos:year

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year
```

```
x = Student("Mike", "Olsen", 2019)
print(x.graduationyear)
```

Añadir métodos

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)
```

Agregue un método llamado a la clase:welcomeStudent

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

Si agrega un método en la clase secundaria con el mismo nombre que una función en la clase primaria, se invalidará la herencia del método primario.

Iteradores de Python

Un iterador es un objeto que contiene un número contable de valores.

Un iterador es un objeto sobre el que se puede iterar, lo que significa que puede recorrer todos los valores.

Técnicamente, en Python, un iterador es un objeto que implementa el protocolo de iterador, que consiste en los métodos `__iter__()` y . `__next__()`

Iterador vs Iterable

Las listas, tuplas, diccionarios y conjuntos son objetos iterables. Son *contenedores* iterables de los que puede obtener un iterador.

Todos estos objetos tienen un método `iter()` que se utiliza para obtener un iterador:

Devuelve un iterador de una tupla e imprime cada valor:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Incluso las cadenas son objetos iterables y pueden devolver un iterador:

Las cadenas también son objetos iterables, que contienen una secuencia de caracteres:

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

Bucle a través de un iterador

También podemos usar un bucle para iterar a través de un objeto iterable:`for`

Iterar los valores de una tupla:

```
mytuple = ("apple", "banana", "cherry")
for x in mytuple:
```

```
  print(x)
```

Iterar los caracteres de una cadena:

```
mystr = "banana"
for x in mystr:
  print(x)
```

El bucle realmente crea un objeto de iterador y ejecuta el método next() para cada bucle.for

Crear un iterador

Para crear un objeto/clase como iterador, debe implementar los métodos y el objeto.__iter__() __next__()

Como ha aprendido en el capítulo Clases/Objetos de Python, todas las clases tienen una función llamada.__init__() , que le permite inicializarse cuando se crea el objeto

El método actúa de forma similar, puede realizar operaciones (inicialización, etc.), pero siempre debe devolver el propio objeto de iterador.__iter__()

El método __next__() también permite realizar operaciones y debe devolver el siguiente elemento de la secuencia.

Cree un iterador que devuelva números, empezando por 1, y cada secuencia aumentará en uno (devolviendo 1,2,3,4,5 etc.):

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
            x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)
```

```
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

StopIteration

El ejemplo anterior continuaría para siempre si tuviera suficientes instrucciones next(), o si se utilizaba en un bucle.for

Para evitar que la iteración se siga para siempre, podemos usar la instrucción.StopIteration

En el método, podemos agregar una condición de terminación para generar un error si la iteración se realiza un número especificado de veces:__next__()

Detener después de 20 iteraciones:

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
            if self.a <= 20:
            x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)
```

```
for x in myiter:
  print(x)
```

Módulos Python

¿Qué es un módulo?

Considere que un módulo es el mismo que una biblioteca de código.

Un archivo que contiene un conjunto de funciones que desea incluir en la aplicación.

Crear un módulo

Para crear un módulo sólo tiene que guardar el código que desea en un archivo con la extensión de archivo :.py

Guarde este código en un archivo denominado mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Usar un módulo

Ahora podemos usar el módulo que acabamos de crear, usando la instrucción:import

Importe el módulo denominado mymodule y llame a la función de saludo:

```
import mymodule
mymodule.greeting("Jonathan")
```

Nota: Cuando utilice una función de un módulo, utilice la sintaxis: *module_name.function_name*.

Variables en el módulo

El módulo puede contener funciones, como ya se ha descrito, pero también variables de todos los tipos (arrays, diccionarios, objetos, etc.)

Guarde este código en el archivo mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Importe el módulo denominado mymodule y acceda al diccionario person1:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Volver a nombrar un módulo

Puede crear un alias al importar un módulo mediante la palabra clave:as

Cree un alias para mymodule llamado: mx

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Módulos incorporados

Hay varios módulos integrados en Python, que puede importar cuando lo desee.

Importe y utilice el módulo:platform

```
import platform

x = platform.system()
print(x)
```

Uso de la función dir()

Hay una función integrada para enumerar todos los nombres de función (o nombres de variables) en un módulo. La función:dir()

Enumere todos los nombres definidos que pertenecen al módulo de plataforma:

```
import platform

x = dir(platform)
print(x)
```

Nota: La función dir() se puede utilizar en *todos los* módulos, también en los que usted mismo crea.

Importar desde módulo

Puede elegir importar solo piezas de un módulo mediante la palabra clave.from

El módulo denominado mymodule tiene una función y un diccionario:

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Importe sólo el diccionario person1 del módulo:

```
from mymodule import person1

print (person1["age"])
```

Nota: Al importar utilizando la palabra clave, no utilice el nombre del módulo al hacer referencia a los elementos del módulo. Ejemplo: , **no** fromperson1["age"] ~~mymodule.person1["age"]~~

Python Datetime

Fechas de Python

Una fecha en Python no es un tipo de datos propio, pero podemos importar un módulo denominado datetime para trabajar con fechas como objetos de fecha.

Importe el módulo datetime y muestre la fecha actual:

```
import datetime

x = datetime.datetime.now()
print(x)
```

Salida de fecha

Cuando ejecutamos el código del ejemplo anterior, el resultado será:

2020-08-12 14:18:57.666217

La fecha contiene año, mes, día, hora, minuto, segundo y microsegundo.

El módulo datetime tiene muchos métodos para devolver información sobre el objeto de fecha.

Devolver el año y el nombre del día de la semana:

```
import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

Creación de objetos de fecha

Para crear una fecha, podemos usar la clase (constructor) del módulo.datetime()datetime

La clase requiere tres parámetros para crear una fecha: año, mes, día.datetime()

Crear un objeto de fecha:

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

La clase también toma parámetros para la hora y la zona horaria (hora, minuto, segundo, microsegundo, tzone), pero son opcionales y tienen un valor predeterminado de , (para la zona horaria).datetime()0None

El método strftime()

El objeto tiene un método para dar formato a los objetos de fecha en cadenas legibles.datetime

Se llama al método strftime(), y toma un parámetro, format , para especificar el formato de la cadena devuelta:

Mostrar el nombre del mes:

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

Matemáticas de Python

Python tiene un conjunto de funciones matemáticas integradas, incluyendo un extenso módulo matemático, que le permite realizar tareas matemáticas en números.

Funciones matemáticas integradas

Las funciones y se pueden utilizar para encontrar el valor más bajo o más alto en un iterable:`min()` `max()`

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

La función devuelve el valor absoluto (positivo) del número especificado:`abs()`

```
x = abs(-7.25)

print(x)
```

La función devuelve el valor de x a la potencia de y (x`pow(x, y)`^{y}).

Devolver el valor de 4 a la potencia de 3 (igual que 4 * 4 * 4):

```
x = pow(4, 3)

print(x)
```

El módulo de matemáticas

Python también tiene un módulo integrado llamado math, que amplía la lista de funciones matemáticas.

Para usarlo, debe importar el módulo:math

```
import math
```

Cuando haya importado el módulo, puede empezar a utilizar métodos y constantes del módulo.math

El método sqrt(), por ejemplo, devuelve la raíz cuadrada de un número:math.

```
import math

x = math.sqrt(64)

print(x)
```

El método math.ceil()redondea un número hacia arriba hasta su entero más cercano y el método math.floor()redondea un número hacia abajo a su entero más cercano y devuelve el resultado:

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```

La constante math.pi, devuelve el valor de PI (3.14...):

```
import math

x = math.pi

print(x)
```

Módulo matemático de Python

Python tiene un módulo integrado que puede usar para tareas matemáticas.

El módulo tiene un conjunto de métodos y constantes.math

Métodos matemáticos

Method	Description
math.acos(x)	Returns the arc cosine value of x
math.acosh(x)	Returns the hyperbolic arc cosine of x
math.asin(x)	Returns the arc sine of x
math.asinh(x)	Returns the hyperbolic arc sine of x
math.atan(x)	Returns the arc tangent value of x
math.atan2(y, x)	Returns the arc tangent of y/x in radians
math.atanh(x)	Returns the hyperbolic arctangent value of x
math.ceil(x)	Rounds a number upwards to the nearest integer, and returns the result
math.comb(n, k)	Returns the number of ways to choose k items from n items without repetition and order
math.copysign(x, y)	Returns a float consisting of the value of the first parameter and the sign of the second parameter
math.cos(x)	Returns the cosine of x
math.cosh(x)	Returns the hyperbolic cosine of x
math.degrees(x)	Converts an angle from radians to degrees

math.dist(p, q)	Calculates the euclidean distance between two specified points (p and q), where p and q are the coordinates of that point
math.erf(x)	Returns the error function of x
math.erfc(x)	Returns the complementary error function of x
math.exp(x)	Returns the value of E^x, where E is Euler's number (approximately 2.718281...), and x is the number passed to it
math.expm1(x)	Returns the value of E^x - 1, where E is Euler's number (approximately 2.718281...), and x is the number passed to it
math.fabs(x)	Returns the absolute value of a number
math.factorial()	Returns the factorial of a number
math.floor(x)	Rounds a number downwards to the nearest integer, and returns the result
math.fmod(x, y)	Returns the remainder of specified numbers when a number is divided by another number
math.frexp()	Returns the mantissa and the exponent, of a specified value
math.fsum(iterable)	Returns the sum of all items in an iterable (tuples, arrays, lists, etc.)
math.gamma(x)	Returns the gamma value of x
math.gcd()	Returns the highest value that can divide two integers
math.hypot()	Find the Euclidean distance from the origin for *n* inputs

math.isclose()	Checks whether two values are close, or not
math.isfinite(x)	Checks whether x is a finite number
math.isinf(x)	Check whether x is a positive or negative infinty
math.isnan(x)	Checks whether x is NaN (not a number)
math.isqrt(n)	Returns the nearest integer square root of n
math.ldexp(x, i)	Returns the expression x * 2i where x is mantissa and i is an exponent
math.lgamma(x)	Returns the log gamma value of x
math.log(x, base)	Returns the natural logarithm of a number, or the logarithm of number to base
math.log10(x)	Returns the base-10 logarithm of x
math.log1p(x)	Returns the natural logarithm of 1+x
math.log2(x)	Returns the base-2 logarithm of x
math.perm(n, k)	Returns the number of ways to choose k items from n items with order and without repetition
math.pow(x, y)	Returns the value of x to the power of y
math.prod(iterable, *, start=1)	Returns the product of an iterable (lists, array, tuples, etc.)
math.radians(x)	Converts a degree value (x) to radians

math.remainder(x, y)	Returns the closest value that can make numerator completely divisible by the denominator
math.sin(x)	Returns the sine of x
math.sinh(x)	Returns the hyperbolic sine of x
math.sqrt(x)	Returns the square root of x
math.tan(x)	Returns the tangent of x
math.tanh(x)	Returns the hyperbolic tangent of x
math.trunc(x)	Returns the truncated integer parts of x

Constantes matemáticas

Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

Fuentes: https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion9/herencia.html
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math