IIC1103 - Sección 2

Clase 20: Programación orientada a objetos 2

Prof. Francisca Cattan

Universidad Católica de Chile

27 de octubre, 2022

Repaso clases

Python nos permite crear nuestras propias clases, es decir clasificaciones originales de valores y métodos.

Repaso clases

Python nos permite crear nuestras propias clases, es decir clasificaciones originales de valores y métodos.
Las clases comienzan siendo definidas con `class Nombre:`

Repaso clases

Python nos permite crear nuestras propias clases, es decir clasificaciones originales de valores y métodos.
Las clases comienzan siendo definidas con `class Nombre:`

- Las clases tienen atributos y métodos (funciones).

Repaso clases

Python nos permite crear nuestras propias clases, es decir clasificaciones originales de valores y métodos.
Las clases comienzan siendo definidas con `class Nombre:`

- Las clases tienen atributos y métodos (funciones).
- Si `x` es un objeto, accedemos a sus elementos como `x.atributo` y `x.metodo(. . .)`

Repaso clases

Python nos permite crear nuestras propias clases, es decir clasificaciones originales de valores y métodos.
Las clases comienzan siendo definidas con `class Nombre:`

- Las clases tienen atributos y métodos (funciones).
- Si `x` es un objeto, accedemos a sus elementos como `x.atributo` y `x.metodo(. . .)`
- Los métodos de una clase siempre reciben como primer parámetro a `self`, que representa al objeto con el que se está trabajando.

Repaso clases

Python nos permite crear nuestras propias clases, es decir clasificaciones originales de valores y métodos.
Las clases comienzan siendo definidas con `class Nombre:`

- Las clases tienen atributos y métodos (funciones).
- Si `x` es un objeto, accedemos a sus elementos como `x.atributo` y `x.metodo(. . .)`
- Los métodos de una clase siempre reciben como primer parámetro a `self`, que representa al objeto con el que se está trabajando.
- Existe un primer método a llamar, **el constructor**, que inicializa los objetos de la clase. La sintaxis de este método es `__init__(self, . . .)`

Repaso clases

Python nos permite crear nuestras propias clases, es decir clasificaciones originales de valores y métodos.
Las clases comienzan siendo definidas con `class Nombre:`

- Las clases tienen atributos y métodos (funciones).
- Si `x` es un objeto, accedemos a sus elementos como `x.atributo` y `x.metodo(. . .)`
- Los métodos de una clase siempre reciben como primer parámetro a `self`, que representa al objeto con el que se está trabajando.
- Existe un primer método a llamar, **el constructor**, que inicializa los objetos de la clase. La sintaxis de este método es `__init__(self, . . .)`
- Cuando se crean objetos, deben pasarse como parámetros o argumentos las variables definidas en `__init__`, excepto por `self`.

Programación orientada a objetos

Creemos varios estudiantes de la clase y agreguemos atributos.

```
1  from random import randint # siempre se importa al inicio
2  class Estudiante:
3      def __init__(self, n, a, i1):
4          print("Voy a crear a {}".format(n))
5          self.nombre = n
6          self.apellido = a
7          self.nota_i1 = i1
8          self.seccion = 2
9
10 estud_1 = Estudiante("Sasha", "Blouse", 6.0)
11 estud_2 = Estudiante("Armin", "Arlert", randint(10,70)/10)
12 estud_3 = Estudiante("Jean", "Kirstein", randint(10,70)/10)
```

Programación orientada a objetos

Creemos varios estudiantes de la clase y agreguemos atributos.

```
1  from random import randint # siempre se importa al inicio
2  class Estudiante:
3      def __init__(self, n, a, i1):
4          print("Voy a crear a {}".format(n))
5          self.nombre = n
6          self.apellido = a
7          self.nota_i1 = i1
8          self.seccion = 2
9
10 estud_1 = Estudiante("Sasha", "Blouse", 6.0)
11 estud_2 = Estudiante("Armin", "Arlert", randint(10,70)/10)
12 estud_3 = Estudiante("Jean", "Kirstein", randint(10,70)/10)
```

```
Voy a crear a Sasha
Voy a crear a Armin
Voy a crear a Jean
```

Programación orientada a objetos

Cada clase tiene su propio **constructor**, método que se llama cuando se crea un nuevo objeto de la clase. Dentro del constructor, **self** se refiere al objeto que esta siendo creado.

```
1  class Estudiante:
2      def __init__(self, n, a, i1):
3          self.nombre = n
4          self.apellido = a
5          self.nota_i1 = i1
6          self.seccion = 2
7
8  class Profesor:
9      def __init__(self, n, a, s):
10         self.nombre = n
11         self.apellido = a
12         self.seccion = s
13
14 estud_1 = Estudiante("Sasha", "Blouse", 6.0)
15 estud_2 = Estudiante("Armin", "Arlert", 6.3)
16 profe = Profesor("Erwin", "Smith", 2)
```

Métodos de clase

Un **método** es una **función** que es parte de una clase.

```
1  class Estudiante:
2      def __init__(self, n, a, i1):
3          self.nombre = n
4          self.apellido = a
5          self.nota_i1 = i1
6          self.seccion = 4
7
8      def imprimir_datos(self):
9          print("La nota de {} es {}".format(self.nombre,self.
   nota_i1))
10
11     def subir_nota(self, decimas):
12         self.nota_i1 += decimas
13         if self.nota_i1 > 7.0:
14             self.nota_i1 = 7.0
```

Métodos de clase

Pensemos juntos: Ahora ambas clases tienen un metodo imprimir_datos(), ¿existe un conflicto con esto?

```
1 class Profesor:
2     def __init__(self, n, a, s):
3         self.nombre = n
4         self.apellido = a
5         self.seccion = s
6
7     def imprimir_datos(self):
8         print("El profesor {} {} es de la seccion {}".
  format(self.nombre,self.apellido,self.seccion))
```

Métodos de clase

EL metodo que se ejecuta depende de la **clase** a la que pertenece el **objeto**.

```
1  b = Estudiante("Francisco", "Diaz", 6.3)
2  c = Estudiante("Catalina", "Huerta", 5.3)
3  d = Estudiante("Pamela", "Leiva", 4.3)
4  p = Profesor("Cristian", "Ruz", 9)
5  v = Profesor("Valeria", "Herskovic", 5)
6
7  c.imprimirDatos()
8  d.subir_nota(0.8)
9  b.imprimirDatos()
10 p.imprimirDatos()
11 v.imprimirDatos()
```

Para b,c,d se llama al imprimir_datos() de la clase Estudiante.
Para p,v se llama a imprimir_datos() de la clase Profesor

Metodo str

- El metodo __str__ nos ayuda a asociar la representación en string del objeto.
- Se llama cuando invocamos las funciones print() o str().
- Al definirlo y usarlo, nos ayuda a controlar cómo es la representación del objeto en el tipo string.

Metodo str

- El metodo __str__ nos ayuda a asociar la representación en string del objeto.
- Se llama cuando invocamos las funciones print() o str().
- Al definirlo y usarlo, nos ayuda a controlar cómo es la representación del objeto en el tipo string.

```
1  class Animal():
2      def __init__(self, animal, especie):
3          self.animal = animal
4          self.especie = especie
5      def __str__(self):
6          mensaje = "Yo soy un {} chilena y mi especie es {}"
   .format(self.animal, self.especie)
7          return mensaje
8  ani = Animal("ave", "Caiquen")
9  print(ani)
10 s = str(ani) # >> 'Yo soy un ave chilena y mi especie es
   Caiquen'
```

Metodo str

- El metodo __str__ nos ayuda a asociar la representación en string del objeto.
- Se llama cuando invocamos las funciones print() o str().
- Al definirlo y usarlo, nos ayuda a controlar cómo es la representación del objeto en el tipo string.

```
1  class Animal():
2      def __init__(self, animal, especie):
3          self.animal = animal
4          self.especie = especie
5      def __str__(self):
6          mensaje = "Yo soy un {} chilena y mi especie es {}"
   .format(self.animal, self.especie)
7          return mensaje
8  ani = Animal("ave", "Caiquen")
9  print(ani)
10 s = str(ani) # >> 'Yo soy un ave chilena y mi especie es
   Caiquen'
```

Yo soy un ave chilena y mi especie es Caiquen

Metodo float

- El metodo __float__ nos ayuda a asociar la representación en float del objeto.
- Se llama cuando invocamos a la función float().
- Al definirlo y usarlo, nos ayuda a controlar cómo es la representación del objeto en el tipo float.

Metodo float

- El metodo __float__ nos ayuda a asociar la representación en float del objeto.
- Se llama cuando invocamos a la función float().
- Al definirlo y usarlo, nos ayuda a controlar cómo es la representación del objeto en el tipo float.

```
1  class Fraccion:
2      def __init__(self, numerador, denominador):
3          self.numerador = numerador
4          self.denominador = denominador
5
6      def __float__(self):
7          return self.numerador / self.denominador
8
9  f1 = Fraccion(10, 5)
10 print('El valor de f1 es', float(f1))
11 f2 = Fraccion(22, 9)
12 print('El valor de f2 es', float(f2))
```

Metodo float

- El metodo __float__ nos ayuda a asociar la representación en float del objeto.
- Se llama cuando invocamos a la función float().
- Al definirlo y usarlo, nos ayuda a controlar cómo es la representación del objeto en el tipo float.

```
1  class Fraccion:
2      def __init__(self, numerador, denominador):
3          self.numerador = numerador
4          self.denominador = denominador
5
6      def __float__(self):
7          return self.numerador / self.denominador
8
9  f1 = Fraccion(10, 5)
10 print('El valor de f1 es', float(f1))
11 f2 = Fraccion(22, 9)
12 print('El valor de f2 es', float(f2))
```

```
El valor de f1 es 2.0
El valor de f2 es 2.444444444446
```

¡Programemos!

