IIC1103 - Sección 2

Clase 21: Interacción entre objetos

Prof. Francisca Cattan

Universidad Católica de Chile

14 de noviembre

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`.

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
```

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto?

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto?

```
1  class Libro:
2    def __init__(self, nombre, autor, genero):
3      self.nombre = nombre
4      self.autor = autor
5      self.genero = genero
6
7  class Biblioteca:
8    def __init__(self):
9      self.cantidad_libros = 0
10     self.lista_libros = []
11     self.lista_autores = []
12
13 # Que cosas podriamos hacer con estas dos clases?
```

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto?

```
1  class Libro:
2    def __init__(self, nombre, autor, genero):
3      self.nombre = nombre
4      self.autor = autor
5      self.genero = genero
6
7  class Biblioteca:
8    def __init__(self):
9      self.cantidad_libros = 0
10     self.lista_libros = []
11     self.lista_autores = []
12
13   def agregar_libro(self, libro):
14     # Como agrego un libro a la lista de libros?
15     # Como agrego el autor del libro a la lista de autores?
16     # Deberia aumentar mi cantidad de libros?
```

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto?

```
1  class Libro:
2    def __init__(self, nombre, autor, genero):
3      self.nombre = nombre
4      self.autor = autor
5      self.genero = genero
6
7  class Biblioteca:
8    def __init__(self):
9      self.cantidad_libros = 0
10     self.lista_libros = []
11     self.lista_autores = []
12
13   def agregar_libro(self, libro):
14     self.lista_libros.append(libro)
15     self.lista_autores.append(libro.autor)
16     self.cantidad_libros += 1
```

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto? Es posible acceder a los atributos y métodos de un objeto en cualquier momento. Eso si, requiere de mucho cuidado y conocimiento de tu código al momento de desarrollar.

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto? Es posible acceder a los atributos y métodos de un objeto en cualquier momento. Eso si, requiere de mucho cuidado y conocimiento de tu código al momento de desarrollar.

```
1  class Estudiante:
2    def __init__(self, nombre, numero_alumno):
3      self.nombre = nombre
4      self.numero_alumno = numero_alumno
5      self.cursos_tomados = []
6
7    def asignar(self, curso_actual):
8      self.cursos_tomados.append(curso_actual)
9      curso_actual.agregar_estudiante(self) # que pasa aqui?
10
11 class CursoActual:
12   def __init__(self, nombre, sigla):
13     self.nombre = nombre
14     self.sigla = sigla
15     self.estudiantes = []
16
17   def agregar_estudiante(self, estudiante):
18     self.estudiantes.append(estudiante)
```

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto? Es posible acceder a los atributos y métodos de un objeto en cualquier momento. Eso si, requiere de mucho cuidado y conocimiento de tu código al momento de desarrollar.

```
1  class Estudiante:
2    def __init__(self, nombre, numero_alumno):
3      self.nombre = nombre
4      self.numero_alumno = numero_alumno
5      self.cursos_tomados = []
6
7    def asignar(self, curso_actual):
8      self.cursos_tomados.append(curso_actual)
9      curso_actual.agregar_estudiante(self) # que pasa aqui?
10
11 class CursoActual:
12   def __init__(self, nombre, sigla):
13     self.nombre = nombre
14     self.sigla = sigla
15     self.estudiantes = []
16
17   def agregar_estudiante(self, estudiante):
18     self.estudiantes.append(estudiante)
```

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto? Es posible acceder a los atributos y métodos de un objeto en cualquier momento. Eso si, requiere de mucho cuidado y conocimiento de tu código al momento de desarrollar.

```
1  class Estudiante:
2    def __init__(self, nombre, numero_alumno):
3      self.nombre = nombre
4      self.numero_alumno = numero_alumno
5      self.cursos_tomados = []
6
7    def asignar(self, curso_actual):
8      self.cursos_tomados.append(curso_actual)
9      curso_actual.agregar_estudiante(self) # que pasa aqui?
10
11 class CursoActual:
12   def __init__(self, nombre, sigla):
13     self.nombre = nombre
14     self.sigla = sigla
15     self.estudiantes = []
16
17   def agregar_estudiante(self, estudiante):
18     self.estudiantes.append(estudiante)
```

Interaccion entre tipos

¿Qué pasaría si yo quiero usar los datos de un objeto en otro objeto? Es posible acceder a los atributos y métodos de un objeto en cualquier momento. Eso si, requiere de mucho cuidado y conocimiento de tu código al momento de desarrollar.

```
1  class Estudiante:
2    def __init__(self, nombre, numero_alumno):
3      self.nombre = nombre
4      self.numero_alumno = numero_alumno
5      self.cursos_tomados = []
6
7    def asignar(self, curso_actual):
8      self.cursos_tomados.append(curso_actual)
9      curso_actual.agregar_estudiante(self) # que pasa aqui?
10
11 class CursoActual:
12   def __init__(self, nombre, sigla):
13     self.nombre = nombre
14     self.sigla = sigla
15     self.estudiantes = []
16
17   def agregar_estudiante(self, estudiante):
18     self.estudiantes.append(estudiante)
```

Interaccion entre tipos

```
1  class Estudiante:
2    def __init__(self, nombre, numero_alumno):
3      self.nombre = nombre
4      self.numero_alumno = numero_alumno
5      self.cursos_tomados = []
6
7    def asignar(self, curso_actual):
8      self.cursos_tomados.append(curso_actual)
9      curso_actual.agregar_estudiante(self) # que pasa aqui?
10
11 class CursoActual:
12   def __init__(self, nombre, sigla):
13     self.nombre = nombre
14     self.sigla = sigla
15     self.estudiantes = []
16
17   def agregar_estudiante(self, estudiante):
18     self.estudiantes.append(estudiante)
```

```
1 curso = CursoActual("Intro a la Progra",'IIC1103')
2 bob = Estudiante("Bob Ross", 21035413)
3 bob.asignar(curso)
4 curso.estudiantes[0].nombre #>> 'Bob Ross'
```