

Constructors and Magic Methods



Today starts as a Paper + Pencil or Tablet + Pencil day... please keep laptops stowed away!

COMP110 - CL20

2024/04/18

Warm-up: Respond to questions on GradeScope and then Diagram

This will be today's attendance submission. Individual submissions!

```
1  class Dog:
2      name: str
3      age: int
4
5      def initialize(self, name: str, age: int) -> None:
6          self.name = name
7          self.age = age
8
9
10  ada: Dog = Dog()
11  ada.initialize("Ada", 5)
12
13  nelli: Dog = Dog()
14  nelli.initialize("Nelli", 11)
15
16  print(nellyi.age)
17  print(ada.age)
```

```
1 class Dog:
2     name: str
3     age: int
4
5     def initialize(self, name: str, age: int) -> None:
6         self.name = name
7         self.age = age
8
9
10    ada: Dog = Dog()
11    ada.initialize("Ada", 5)
12
13    nelli: Dog = Dog()
14    nelli.initialize("Nelli", 11)
15
16    print(nelly.age)
17    print(ada.age)
```

Notes on Constructors and the `__init__` Method

Example of `__init__`

The opening warm-up diagram could be idiomatically achieved as follows

```
1 class Dog:
2     name: str
3     age: int
4
5     def intialize(self, name: str, age: int) -> None:
6         self.name = name
7         self.age = age
8
9
10 ada: Dog = Dog()
11 ada.intialize("Ada", 5)
12
13 nelli: Dog = Dog()
14 nelli.intialize("Nelli", 11)
15
16 print(nellyi.age)
17 print(ada.age)
```

Original

```
1 class Dog:
2     name: str
3     age: int
4
5     def __init__(self, name: str, age: int):
6         self.name = name
7         self.age = age
8
9
10 ada: Dog = Dog("Ada", 5)
11 nelli: Dog = Dog("Nelli", 11)
12 print(nellyi.age)
13 print(ada.age)
```

Special method called during construction. (Note: Two underscores on *both* sides.)

Notice in the *constructor call*, we are passing arguments to the special `__init__` method.

Idiomatic with `__init__` for Construction

Diagram the code listing

```
1 class Dog:
2     name: str
3
4     def __init__(self, name: str):
5         print("Dog#__init__")
6         self.name = name
7
8     def speak(self) -> str:
9         return f"{self.name}: WOOF"
10
11
12 class Cat:
13     name: str
14
15     def __init__(self, name: str):
16         print("Cat#__init__")
17         self.name = name
18
19     def speak(self) -> str:
20         return f"{self.name}: MEOW"
21
22
23 a: Cat = Cat("Hank")
24 b: Dog = Dog("Boots")
25 print(b.speak())
26 print(a.speak())
```

```
1 class Dog:
2     name: str
3
4     def __init__(self, name: str):
5         print("Dog#__init__")
6         self.name = name
7
8     def speak(self) -> str:
9         return f"{self.name}: WOOF"
10
11
12 class Cat:
13     name: str
14
15     def __init__(self, name: str):
16         print("Cat#__init__")
17         self.name = name
18
19     def speak(self) -> str:
20         return f"{self.name}: MEOW"
21
22
23 a: Cat = Cat("Hank")
24 b: Dog = Dog("Boots")
25 print(b.speak())
26 print(a.speak())
```

Diagram the code listing

```
1 from typing import Self
2
3
4 class Point:
5     x: float
6     y: float
7
8     def __init__(self, x: float, y: float):
9         self.x = x
10        self.y = y
11
12    def distance(self, to: Self) -> float:
13        d_x2: float = (self.x - to.x) ** 2
14        d_y2: float = (self.y - to.y) ** 2
15        return (d_x2 + d_y2) ** 0.5
16
17
18 a: Point = Point(1.0, 1.0)
19 b: Point = Point(1.0, 3.0)
20 print(b.distance(a))
```


Code Follow Along

Making use of `__str__` and `__repr__` Magic "Dunder" Methods

Motivations for Classes and OOP

Modeling and Abstraction

Encapsulation

Modularity