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

Constructors and Magic Methods

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	<pre>def intialize(self, name: str, age</pre>
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(nelli.age)
17	<pre>print(ada.age)</pre>

e: int) -> None:

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

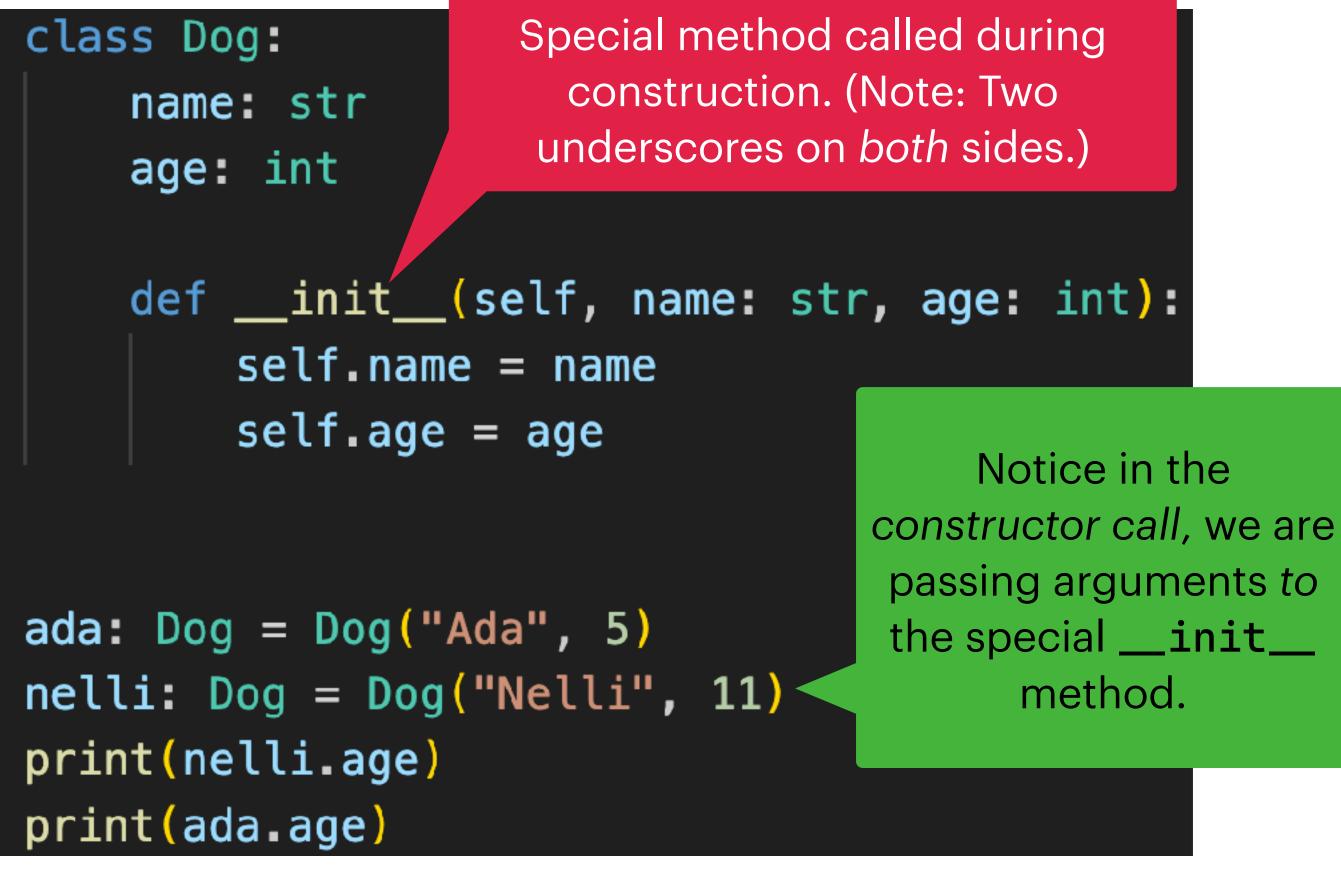
Notes on Constructors and the _____init____Method

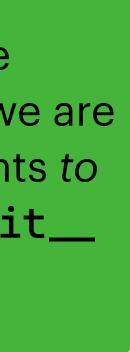
Example of _____init

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

1	class Dog:
2	name: str
3	age: int
4	
5	<pre>def intialize(self, name: str, age: int) -> None:</pre>
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(nelli.age)
17	print(ada.age)

Original





```
class Dog:
 1
 2
          name: str
 3
 4
          def __init__(self, name: str):
 5
              print("Dog#___init___")
 6
              self_name = name
 7
 8
          def speak(self) -> str:
              return f"{self.name}: WOOF"
 9
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:
              return f"{self.name}: MEOW"
20
21
22
      a: Cat = Cat("Hank")
23
      b: Dog = Dog("Boots")
24
      print(b.speak())
25
      print(a.speak())
26
```

Diagram the code listing

```
class Dog:
 1
 2
          name: str
 3
 4
          def ___init__(self, name: str):
 5
              print("Dog#___init___")
              self.name = name
 6
 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
      a: Cat = Cat("Hank")
23
      b: Dog = Dog("Boots")
24
      print(b.speak())
25
      print(a.speak())
26
```

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

Diagram the code listing



Code Follow Along Making use of _str_ and _repr_ Magic "Dunder" Methods

Motivations for Classes and OOP

Modeling and Abstraction

Encapsulation

Modularity