[320] Special Methods

Yiyin Shen

Classes

```
class Dog:
    def init(dog):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog()
```

which one is an attribute?

1. dog
2. name
3. mult
4. fido

Classes

```
class Dog:
    def init(dog):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog()
```

is this printed? do we crash?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)
```

is this printed? do we crash?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)
```

```
speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which call won't work?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which call won't work?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which one is NOT an example of type-based dispatch?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which one is NOT an example of type-based dispatch?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which call style is preferred?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

fido.speak(5)
```

preferred style

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)


fido.speak(5)
```

what will be passed to the dog param?

Classes

what is a better name for the receiver parameter?

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)


fido.speak(5)
```

Classes

what is a better name for the receiver parameter?

answer: `self`

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)


fido.speak(5)
```

Special Methods

Special Methods

__init__ is a special method, with non-standard behavior

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)


fido.speak(5)
```

Special Methods

There are MANY special method names:
https://docs.python.org/3/reference/datamodel.html#special-method-names

We'll learn a few:

`__str__, __repr__, _repr_html_`

`__eq__, __lt__`

`__len__, __getitem__`

`__enter__, __exit__`

control how an object looks when we print it or see it in Out[N]

generate HTML to create more visual representations of objects in Jupyter. Like tables for DataFrames

Special Methods

There are MANY special method names:
https://docs.python.org/3/reference/datamodel.html#special-method-names

We'll learn a few:

`__str__, __repr__, _repr_html_`

`__eq__, __lt__`

define how **==** behaves for two different objects

`__len__, __getitem__`

define how a list of objects should be sorted

`__enter__, __exit__`

```
c = (a==b) # type of c?
```

Special Methods

There are MANY special method names:
https://docs.python.org/3/reference/datamodel.html#special-method-names

We'll learn a few:

`__str__, __repr__, _repr_html_`

`__eq__, __lt__`

`__len__, __getitem__`

build our own sequences that we index, slice, and loop over:

```
val = obj[idx]
vals = obj[3:7]
for x in obj:
    print(x)
```

what goes in brackets?

`__enter__, __exit__`

Special Methods

There are MANY special method names:
https://docs.python.org/3/reference/datamodel.html#special-method-names

We'll learn a few:

```
__str__, __repr__, _repr_html_

__eq__, __lt__

__len__, __getitem__

__enter__, __exit__
```

context managers

```
with open("file.txt") as f:
    data = f.read()
# automatically close
```