[320] Special Methods

Tyler Caraza-Harter

Midterm Schedule

Date: Wednesday, March 11th

Time: 7:15 pm

Length: 2 hours

Room: Ingraham B10

Conflict? Fill this:

https://forms.gle/Y9xfkBVFy1wgytDu9

McBurney Exam: Thursday, March 12th @ 5:30pm

Review Classes

CLASSES AND OTHER TYPES

https://www.macys.com/shop/product/martha-stewart-collection-set-of-6-cookie-cutters-created-for-macys?ID=5467270

OBJECTS

Review 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

Review 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?

Review 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?

Review 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)                # A
fido.speak(5)                 # B
Dog.speak(fido, 5)            # C
type(fido).speak(fido, 5) # D
```

which call won't work?

Review 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)   # A~~

```
fido.speak(5)                   # B
Dog.speak(fido, 5)              # C
type(fido).speak(fido, 5)       # D
```

which call won't work?

Review 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)              # A~~
fido.speak(5)                 # B
Dog.speak(fido, 5)            # C
type(fido).speak(fido, 5)     # D
```

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

Review 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)                  # A~~
fido.speak(5)                    # B
~~Dog.speak(fido, 5)              # C~~
type(fido).speak(fido, 5)  # D
```

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

Review 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)              # A~~
fido.speak(5)                 # B
~~Dog.speak(fido, 5)          # C~~
type(fido).speak(fido, 5)     # D
```

which call style is preferred?

Review 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)                   # B
```

preferred style

Review 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)                  # B
```

what will be passed to the dog param?

Review 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)                    # B
```

Review 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)                   # B
```

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)                    # B
```

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__`

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)
```

`__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
```

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__
```

example 1: dogs

```
__len__, __getitem__

__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__` example 2: range(...)

`__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__` **example 3:** make our own Series

`__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__`

example 4: plots inside a "with" block will have extra large font

Demos