

Static and Class Methods

Welcome back to CS 2100!

Prof. Rasika Bhalerao

Warm-up exercise: let's write a class `Vector3D`

- Attributes:
 - `self.x: float`
 - `self.y: float`
 - `self.z: float`
- Properties:
 - `self.length: float` (no setter)
 - `self.normalized: 'Vector3D'` (no setter)
- Methods:
 - `self.add(other: 'Vector3D') -> 'Vector3D'`
 - `self.dot_product(other: 'Vector3D') -> float`

Operating on a class rather than an instance

What if we don't have `self` as the first arg?

- `self` gives a method access to instance variables (which is why they start with `self.`)

Instance variables are specific to an instance of the class.

Class variables are shared among the entire class.

- Access class variables using the name of the class (`OnlineStore.base_url`)

Example: class variable `count`

```
class Counter:
    count: int = 0 # count the instances that have been instantiated

    def __init__(self) -> None:
        Counter.count += 1 # increment class counter

ct1 = Counter()

print(ct1.count) # 1

ct2 = Counter()

print(ct1.count) # 2
print(ct2.count) # same as ct1.count because count is a class variable

for i in range(10):
    Counter()

print(ct1.count) # 12
```

Class method: a method which is shared among the entire class

- Must be decorated with `@classmethod`
- First arg must be `cls` (instead of `self`)
 - access class variables from within class methods using `cls`
 - (While we could access them using the name of the class, using `cls` ensures that the class method still works in its subclasses.)

Example: class method to keep track of an API's base URL

If the API's base URL changes, it should change everywhere that it is used.

```
class APIClient:
    base_url = 'https://api.example.com'
    timeout = 30

    @classmethod
    def configure(cls,
                  base_url: Optional[str] = None,
                  timeout: Optional[int] = None
    ) -> None:
        if base_url:
            cls.base_url = base_url
        if timeout:
            cls.timeout = timeout

    @classmethod
    def reset_config(cls) -> None:
        cls.base_url = 'https://api.example.com'
        cls.timeout = 30
```

```
print(APIClient.base_url) # https://api.example.com
print(APIClient.timeout) # 30

APIClient.configure('new_url.com', 60)

user1 = APIClient()
print(user1.base_url) # new_url.com
print(user1.timeout) # 60

APIClient.reset_config()

user2 = APIClient()
print(user2.base_url) # https://api.example.com
print(user2.timeout) # 30
```

Common use of class methods: alternate constructors

```
from datetime import datetime
from typing import TypeVar

T = TypeVar('T', bound='Person') # Generic type that must be a subclass of Person

class Person:
    def __init__(self, name: str, birth_year: int):
        self.name = name
        self.birth_year = birth_year

    @classmethod
    def from_birth_date(cls: type[T], name: str, birth_date_str: str) -> T:
        year = datetime.strptime(birth_date_str, "%Y-%m-%d").year
        return cls(name, year)

    @classmethod
    def baby(cls: type[T], name: str) -> T:
        return cls(name, datetime.now().year)

person1 = Person('Mini', 2015)
person2 = Person.from_birth_date('Binnie', "2020-03-15")
person3 = Person.baby('Ginnie')
```

```

from typing import TypeVar

T = TypeVar('T', bound='Vehicle')

class Vehicle:
    total_vehicles = 0

    def __init__(self, make: str, model: str):
        self.make = make
        self.model = model
        Vehicle.total_vehicles += 1

    @classmethod
    def from_string(
        cls: type[T],
        vehicle_str: str) -> T:
        make, model = vehicle_str.split('-')
        return cls(make, model)

    @classmethod
    def get_total(cls) -> int:
        return cls.total_vehicles

```

```

car1 = Vehicle.from_string("Toyota-Camry")
car2 = Vehicle.from_string("Honda-Accord")

print(Vehicle.get_total())

```

Poll: What will `Vehicle.get_total()` return after the two lines at the end are executed?

1. 0

2. 2

3. 2 (but only if we change

`cls.total_vehicles` to

`Vehicle.total_vehicles` in

`get_total()`)

4. Nothing -- it'll raise an error

`@staticmethod`: slightly different from `@classmethod`

Use `@staticmethod` when:

- The function belongs in the class because it logically fits there, and encapsulation dictates that it belongs there
- But it doesn't need access to the class through the `cls` argument
 - (The code would work if it was a function completely external to the class)

Example: let's add to `Vector3D`:

```
T = TypeVar('T', bound='Vector3D')
```

- `@classmethod`
`def zero(cls: type[T]) -> T` (returns the zero vector)
- `@staticmethod`
`def are_perpendicular(v1: 'Vector3D', v2: 'Vector3D') -> bool`

	@classmethod	@staticmethod
Purpose	Operations that work on the whole class	Independent functions that logically go with the class
Access to instance variables	No	No
Access to class variables	Yes	No
First argument	cls	No requirements

```

from typing import TypeVar

T = TypeVar('T', bound='Shape')

class Shape:
    default_color = 'blue'

    @classmethod
    def create_with_default_color(
        cls: type[T], size: int) -> T:
        return cls(size, cls.default_color)

    @staticmethod
    def calculate_area(
        length: int, width: int) -> int:
        return length * width

    def __init__(
        self, size: int, color: str):
        self.size = size
        self.color = color

class Rectangle(Shape):
    default_color = 'red'

```

Poll: Consider this code:

```

rect = Rectangle.create_with_default_color(10)
area = Rectangle.calculate_area(5, 8)

```

What will be the values of `rect.color` and `area`, using the two variables declared at the end?

1. `rect.color = 'blue'`, `area = 40`
2. `rect.color = 'red'`, `area = 40`
3. `rect.color = 'red'`, `area = 13`
4. Both will raise an error because they're called on a subclass

Poll:

- 1. What is your main takeaway from today?**
- 2. What would you like to revisit next time?**