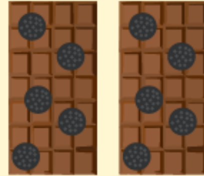
Objects + Classes

Motivation

Building a chocolate shop

Name: Trufflapagus
Price: $9.99
Nutrition: 170 cals, 19 g sugar
Inventory: 2 bars



Name: Piña Chocolotta
Price: $7.99
Nutrition: 200 cals, 24 g sugar
Inventory: 3 bars

Order #1
Visa

Order #2
Discover

Order #3
AmEx

Name: Coco Lover
Address: 123 Pining St
Nibbsville, OH

Name: Nomandy Noms
Address: 34 Slurpalot Pl
Buttertown, IN

Name: Ammar Chako
Address: 42 Milky Way
Temperville, NV

Building a chocolate shop

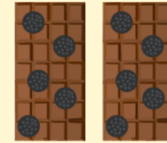
We *could* make data abstractions using functions:

```
# Inventory tracking
add_product(name, price, nutrition)
get_label(product)
get_nutrition_info(product)
increase_inventory(product, amount)
reduce_inventory(product, amount)


# Customer tracking
signup_customer(name, address)
get_greeting(customer)
get_formatted_address(customer)


# Purchase tracking
order(customer, product, quantity, cc_info)
track(order_number)
refund(order_number, reason)
```

Name: Trufflapagus
Price: $9.99
Nutrition: 170 cals, 19 g sugar
Inventory: 2 bars

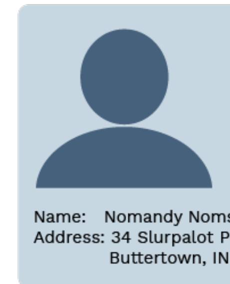


Name: Piña Chocolotta
Price: $7.99
Nutrition: 200 cals, 24 g sugar
Inventory: 3 bars

Order #1
Visa

Order #2
Discover

Order #3
AmEx



That codebase would be organized around functions.

Objects

From functions to objects

We can instead organize around objects:

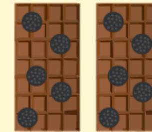
```
# Inventory tracking
Product(name, price, nutrition)
Product.get_label()
Product.get_nutrition_info()
Product.increase_inventory(amount)
Product.reduce_inventory(amount)
Product.get_inventory_report()

# Customer tracking
Customer(name, address)
Customer.get_greeting()
Customer.get_formatted_address()
Customer.buy(product, quantity, cc_info)

# Purchase tracking
Order(customer, product, quantity, cc_info)
Order.ship()
Order.refund(reason)
```

Product

Name: Trufflapagus
Price: $9.99
Nutrition: 170 cals, 19 g sugar
Inventory: 2 bars



Product

Name: Piña Chocolotta
Price: $7.99
Nutrition: 200 cals, 24 g sugar
Inventory: 3 bars

Order

ID: #1
CC: Visa

Order

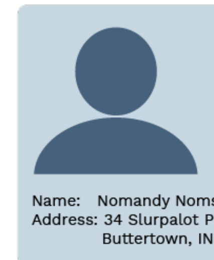
ID: #2
CC: MC

Order

ID: #3
CC: AmEx

Customer

Customer



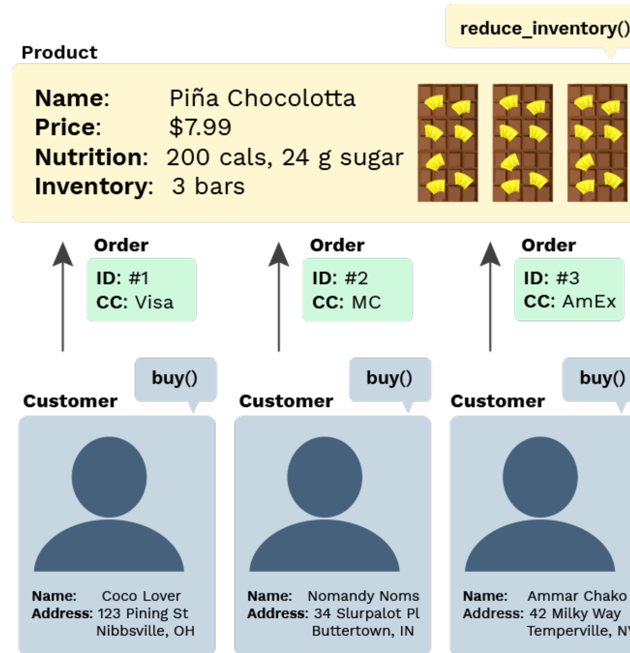
Customer

An object bundles together information and related

behavior.

Concepts

- A **class** is a template for defining new data types.
- An instance of a class is called an **object**.
- Each object has data attributes called **instance variables** that describe its state.
- Each object also has function attributes called **methods**.



Python includes special syntax to create classes and objects.

Classes

What's in a class?

A class can:

- Set the **initial values** for instance variables.
- Define **methods** specific to the object, often used to change or report the values of instance variables.

```
class Product:

    # Set the initial values
    # Define methods
```

Let's code a class!

A fully coded class and usage

```
# Define a new type of data
class Product:

    # Set the initial values
    def __init__(self, name, price, nutrition_info):
        self._name = name
        self._price = price
        self._nutrition_info = nutrition_info
        self._inventory = 0

    # Define methods
    def increase_inventory(self, amount):
        self._inventory += amount

    def reduce_inventory(self, amount):
        self._inventory -= amount

    def get_label(self):
        return "Foxolate Shop: " + self._name

    def get_inventory_report(self):
        if self._inventory == 0:
            return "There are no bars!"
        return f"There are {self._inventory} bars."
```

```
pina_bar = Product("Piña Chocolotta", 7.99,
    ["200 calories", "24 g sugar"])

pina_bar.increase_inventory(2)
```

Let's break it down...

Class definition

```
class Product:

    def __init__(self, name, price, nutrition_info):
    def increase_inventory(self, amount):
    def reduce_inventory(self, amount):
    def get_label(self):
    def get_inventory_report(self):
```

- A class statement creates a new class and binds that class to the class name in the first frame of the current environment.
- Inner `def` statements create attributes of the class (*not* names in frames).

Class instantiation (Object construction)

```
pina_bar = Product("Piña Chocolotta", 7.99,
    ["200 calories", "24 g sugar"])
```

`Product(args)` is often called the **constructor**.

When the constructor is called:

- A new instance of that class is created
- The `__init__` method of the class is called with the new object as its first argument (named `self`), along with any additional arguments provided in the call expression

```
class Product:

    def __init__(self, name, price, nutrition_info):
        self._name = name
        self._price = price
        self._nutrition_info = nutrition_info
        self._inventory = 0
```

Dot notation

All object attributes (which includes variables *and* methods) can be accessed with **dot notation**:

```
pina_bar.increase_inventory(2)
```

That evaluates to the value of the attribute looked up by `increase_inventory` in the object referenced by `pina_bar`.

The left-hand side of the dot notation can also be any expression that evaluates to an object reference:

```
bars = [pina_bar, truffle_bar]
bars[0].increase_inventory(2)
```

Instance variables

Instance variables are data attributes that describe the state of an object.

This `__init__` initializes 4 instance variables:

```
class Product:

    def __init__(self, name, price, nutrition_info):
        self._name = name
        self._price = price
        self._nutrition_info = nutrition_info
        self._inventory = 0
```

The object's methods can then change the values of those variables or assign new variables.

Visualize in PythonTutor

Method invocation

```
pina_bar.increase_inventory(2)
```

```
class Product:
    def increase_inventory(self, amount):
        self._inventory += amount
```

`pina_bar.increase_inventory` is a **bound method**: a function which has its first parameter pre-bound to a particular value.

In this case, `self` is pre-bound to `pina_bar` and `amount` is set to 2.

It's equivalent to:

```
Product.increase_inventory(pina_bar, 2)
```

More on attributes

Dynamic instance variables

An object can create a new instance variable whenever it'd like.

```
class Product:

    def reduce_inventory(self, amount):
        if (self._inventory - amount) <= 0:
            self._needs_restocking = True
        self._inventory -= amount

pina_bar = Product("Piña Chocolotta", 7.99,
    ["200 calories", "24 g sugar"])
pina_bar.reduce_inventory(1)
```

Now `pina_bar` has an updated binding for `_inventory` and a new binding for `_needs_restocking` (which was not in `__init__`).

Visualize in PythonTutor

Class variables

A **class variable** is an assignment inside the class that isn't inside a method body.

```
class Product:
    sales_tax = 0.07
```

Class variables are "shared" across all instances of a class because they are attributes of the class, not the instance.

```
class Product:
    _sales_tax = 0.07

    def get_total_price(self, quantity):
        return (self._price * (1 + self._sales_tax)) * quantity

pina_bar = Product("Piña Chocolotta", 7.99,
    ["200 calories", "24 g sugar"])
truffle_bar = Product("Truffalapagus", 9.99,
    ["170 calories", "19 g sugar"])

pina_bar._sales_tax
truffle_bar._sales_tax
pina_bar.get_total_price(4)
truffle_bar.get_total_price(4)
```

Attributes are all public

As long as you have a reference to an object, you can set or mutate any attributes.

```
pina_bar = Product("Piña Chocolotta", 7.99,
    ["200 calories", "24 g sugar"])

pina_bar._inventory
pina_bar._inventory = 5000000
pina_bar._inventory = -5000
```

You can even assign new instance variables:

```
pina_bar.brand_new_attribute_haha = "instanception"
```

"Private" attributes

To communicate the desired access level of attributes, Python programmers generally use this convention:

- `__` (double underscore) before very private attribute names
- `_` (single underscore) before semi-private attribute names
- no underscore before public attribute names

That allows classes to hide implementation details and add additional error checking.

We will discuss `__` vs `_` next time.

For now, if you see no underscore, HAVE FUN! 🎉🎉

Quiz: Objects + Classes

Multiple instances

There can be multiple instances of each class.

```
pina_bar = Product("Piña Chocolotta", 7.99,
    ["200 calories", "24 g sugar"])

cust1 = Customer("Coco Lover",
    ["123 Pining St", "Nibbsville", "OH"])

cust2 = Customer("Nomandy Noms",
    ["34 Shlurpalot St", "Buttertown", "IN"])
```

What are the classes here?
How many instances of each?

Multiple instances

There can be multiple instances of each class.

```
pina_bar = Product("Piña Chocolotta", 7.99,
    ["200 calories", "24 g sugar"])

cust1 = Customer("Coco Lover",
    ["123 Pining St", "Nibbsville", "OH"])

cust2 = Customer("Nomandy Noms",
    ["34 Shlurpalot St", "Buttertown", "IN"])
```

What are the classes here? `Product`, `Customer`
How many instances of each? 1 `Product`, 2 `Customer`

State management

An object can use instance variables to describe its state. A best practice is to hide the representation of the state and manage it entirely via method calls.

```
>>> pina_bar = Product("Piña Chocolotta", 7.99,
        ["200 calories", "24 g sugar"])

>>> pina_bar.get_inventory_report()
"There are NO bars!"

>>> pina_bar.increase_inventory(3)
>>> pina_bar.get_inventory_report()
"There are 3 bars total (worth $23.97 total)."
```

Product

Name: Piña Chocolotta
Price: $7.99
Nutrition: 200 cals, 24 g sugar
Inventory: 0 bars

Product

Name: Piña Chocolotta
Price: $7.99
Nutrition: 200 cals, 24 g sugar
Inventory: 3 bars

What's the initial state?
What changes the state?

State management

An object can use instance variables to describe its state. A best practice is to hide the representation of the state and manage it entirely via method calls.

```
>>> pina_bar = Product("Piña Chocolotta", 7.99,
        ["200 calories", "24 g sugar"])

>>> pina_bar.get_inventory_report()
"There are NO bars!"

>>> pina_bar.increase_inventory(3)
>>> pina_bar.get_inventory_report()
"There are 3 total (worth $23.97 total)."
```

Product

Name: Piña Chocolotta
Price: $7.99
Nutrition: 200 cals, 24 g sugar
Inventory: 0 bars

Product

Name: Piña Chocolotta
Price: $7.99
Nutrition: 200 cals, 24 g sugar
Inventory: 3 bars

What's the initial state? 0 bars in inventory
What changes the state? `increase_inventory()` by changing the instance variable `_inventory`

Class vs. instance variables

```
class Customer:

    _salutation = "Dear"

    def __init__(self, name, address):
        self._name = name
        self._address = address

    def get_greeting(self):
        return f"{self._salutation} {self._name},"

    def get_formatted_address(self):
        return "\n".join(self._address)

cust1 = Customer("Coco Lover",
    ["123 Pining St", "Nibbsville", "OH"])
```

What are the class variables?

What are the instance variables?

Class vs. instance variables

```
class Customer:

    _salutation = "Dear"

    def __init__(self, name, address):
        self._name = name
        self._address = address

    def get_greeting(self):
        return f"{self._salutation} {self._name},"

    def get_formatted_address(self):
        return "\n".join(self._address)

cust1 = Customer("Coco Lover",
    ["123 Pining St", "Nibbsville", "OH"])
```

What are the class variables? `_salutation`
What are the instance variables? `_name`, `_address`