Practice with basic Classes and Objects in Python

Follow-along #0: Construct a Pizza Object

- Create a file named ls34_object_practice.py
- Establish a Pizza class and main function boilerplate as shown left.
- In the main function:
    1. Declare a variable and assign it a Pizza object. Print this object's size.
    2. Assign different values to each of its three attributes (extra_cheese, toppings). After doing so, print the object's # of toppings again.

```python
"""A demonstration of classes/objects."""


class Pizza:
    """A simple model of a Pizza."""
    size: str = "medium"
    extra_cheese: bool = False
    toppings: int = 0


def main() -> None:
    """Entrypoint of program."""
    ...


if __name__ == "__main__":
    main()
```

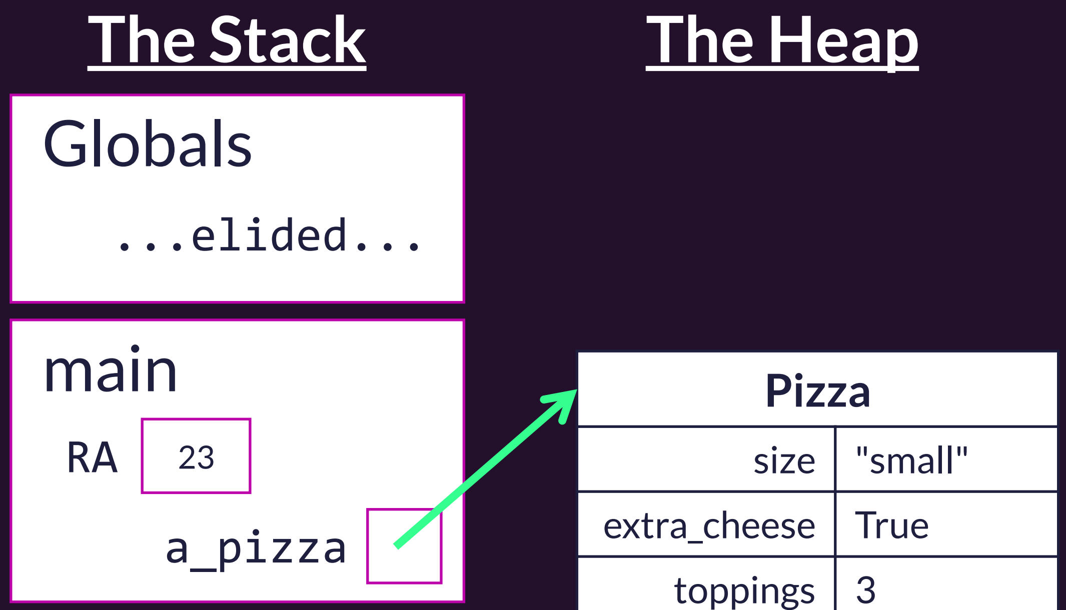
```
# 1. Initialize a variable that holds a Pizza object and print it
a_pizza: Pizza = Pizza()
print(a_pizza.size)

// 2. Assign different values to each of its properties
a_pizza.size = "small";
a_pizza.extraCheese = true;
a_pizza.toppings = 2;
print(str(a_pizza.size) + " with " + str(a_pizza.toppings) + " toppings")
```

Object Values Live on the Heap

Like Lists, objects are *reference types* and *typically mutable*. Their variable names on the call stack hold references to their *actual values* in the heap.

```python
def main() -> None:
    """Entrypoint of program."""
    a_pizza: Pizza = Pizza()
    a_pizza.size = "small"
    a_pizza.extra_cheese = True
    a_pizza.toppings = 3
    print("Size: " + str(a_pizza.size))
    print("EC: " + str(a_pizza.extra_cheese))
    print("Toppings: " + str(a_pizza.toppings))
```



Hands-on: Calculate the Price of a Pizza

3. Declare a **`price`** function that takes a Pizza as a Parameter and returns a float.

4. Correctly implement the **`price function`**:
   - Size sets a base price of $7 small, $9 medium, $11 large
   - Extra cheese adds $1
   - Each topping costs $0.75

5. Call your price function from main and print its result.
   Is it working?

ALWAYS Initialize your Variables

Especially important with variables holding references to objects

- **Example:**

```
pizza1: Pizza
pizza1.size = "large"
> NameError: name 'a_pizza' is not defined
```

- **The fix:** `pizza1: Pizza = Pizza() # Always initialize!`

The "Bundling" of Related Values is an Important Benefit of Objects

- Consider the following two function signatures...

```
def price(size: str, extra_cheese: bool, toppings: int) -> float:

def price(pizza: Pizza) -> float:
```

- Notice with a Pizza data type the function's *semantics* are improved
    - Is the first function calculating the price of a cheeseburger?
    - The second function's signature reads more meaningfully...
      "`price` is a function that is given a `Pizza` object and returns a `number`"

- Consider an object with *far more* properties...
    - Pizza: Base sauce, gluten free crust, thin vs. deep dish, ...
    - Objects give us a convenient means for tightly packaging related variables together