Classes and Objects Practice

Follow-along #0: Construct a Pizza Object

- Before you begin coding, let's establish a Pizza class

- In ls33_pizza_class.ts's 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.

```
"""Another demonstration of classes and objects."""

class Pizza:
    """A simple model of Pizzas."""
    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()
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(f"{a_pizza.size} with {a_pizza.toppings} toppings")
```

Hands-on #1: 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

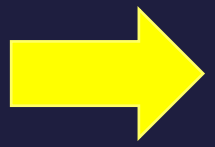
5. Call your price function from main and print its result. Is it working?

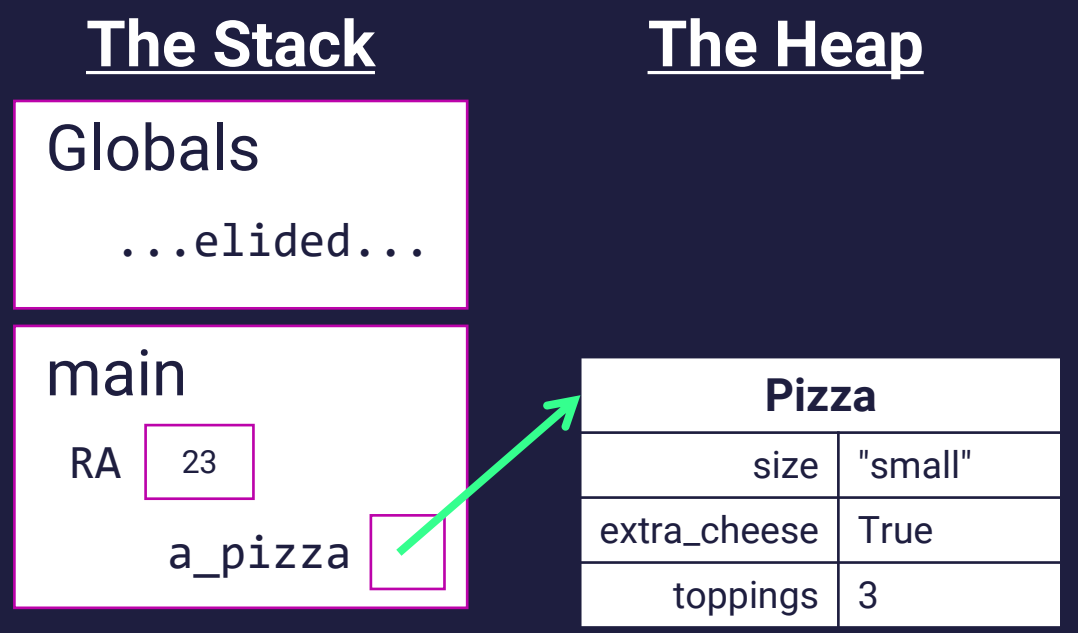
- Check-in on PollEv.com/compunc once your pizza price is correctly calculating! Try changing property values to inspect.

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.

```
11  def main() -> None:
12      """Entrypoint of program."""
13      a_pizza: Pizza = Pizza()
14      a_pizza.size = "small"
15      a_pizza.extra_cheese = True
16      a_pizza.toppings = 3
17      print(f"Size: {a_pizza.size}")
18      print(f"EC: {a_pizza.extra_cheese}")
19      print(f"Toppings: {a_pizza.toppings}")
20
21
22  if __name__ == "__main__":
23      main()
```





ALWAYS Initialize your Variables

Common Error:

```
NameError: name 'a_pizza' is not defined
```

- **Example:**

```
pizza1: Pizza
pizza1.size = "large" # ERROR
```

- **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

```python
"""A diagram with classes and objects."""


class Point:
    """A cartesian coordinate."""
    x: float = 0.0
    y: float = 0.0


def main() -> None:
    """Entrypoint of program."""
    a: Point = Point()
    b: Point = a
    a.x = 4.0

    c: Point = clone(a)
    a.x = 9.0

    print(a.x)
    print(b.x)
    print(c.x)


def clone(p: Point) -> Point:
    """Clone a new Point instance."""
    copy: Point = Point()
    copy.x = p.x
    copy.y = p.y
    return copy


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

Challenge Question. Draw an environment diagram of the code listing and respond to the sequence of questions on PollEverywhere once completed.