`if __name__ == “__main__”` / `chain()` / f-string / Enumerate

If __name__ == “__main__”:

Think of `if __name__ == "__main__"` like a switch that helps python decide:

```
# Let's say this file is called pizza.py

def make_pizza():
    print("Making a pizza!")
    print("Adding cheese")
    print("Adding toppings")

# This is like a switch that only turns on when you run pizza.py directly
if __name__ == "__main__":
    print("Starting the pizza shop!")
    make_pizza()
```

Now, there are two ways to use this code:

1. Run the file directly(`python pizza.py`)

   ```
   Output:
   Starting the pizza shop!
   Making a pizza!
   Adding cheese
   Adding toppings
   ```

2. Import it in another file.

   ```
   # another_file.py
   from pizza import make_pizza

   # Only the function runs, no "Starting the pizza shop!" message
   make_pizza()
   ```

   ```
   Output:
   Making a pizza!
   Adding cheese
   Adding toppings
   ```

Why is this useful?

1. You can test your code by running the file directly

2. You can use your functions in other files without running the test code
3. It keeps your code organized and clean.

Think of it like a recipe book:

- The functions are like the recipes
- The `if __name__ == "__main__":` part is like testing the recipes yourself
- When someone borrows your recipe book, they just want the recipes, not your test notes!

Chain() - Combine multiple Iterables

```
from itertools import chain
list(chain([1, 2], [3, 4]))  # [1, 2, 3, 4]
```

F-strings

Think of f-strings as a way to "fill in the blanks" in your text, where the blanks (curly braces) get replaced with the actual values of your variables.

```
name = "Alice"
age = 25

# Basic f-string
print(f"Hello, my name is {name} and I am {age} years old")
# Output: Hello, my name is Alice and I am 25 years old

# You can do calculations inside f-strings
price = 10
quantity = 3
print(f"Total cost: ${price * quantity}")
# Output: Total cost: $30

# You can format numbers
pi = 3.14159
print(f"Pi rounded to 2 decimals: {pi:.2f}")
# Output: Pi rounded to 2 decimals: 3.14
```

Enumerate

Think of `enumerate()` as a way to count items in a list while you're going through them. It adds a counter to each item in your list, like numbering items on a shopping list.

Here's a simple example:

```
fruits = ["apple", "banana", "cherry"]
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
```

Output:

```
0: apple
1: banana
2: cherry
```

Some key points:

1. By default, it starts counting from 0
2. You can start from a different number if you want:

```
# Start counting from 1 instead of 0
for index, fruit in enumerate(fruits, start=1):
    print(f"{index}: {fruit}")
```

Output:

```
1: apple
2: banana
3: cherry
```