## Different units of code in python

Here's a breakdown of the units of code in Python, typically from smallest to largest in terms of complexity and scope:

1. **Expression**
    - **What it is:** The most fundamental unit of code. An expression is a combination of values, variables, operators, and function calls that **evaluates to a single value**.
    - **Execution Point of View:** When Python encounters an expression, its primary job is to *calculate* or *determine* the value that the expression represents.
    - **Examples:**
        - 10 (a literal value)
        - x (a variable, evaluates to its stored value)
        - 5 + 3 (arithmetic operation, evaluates to 8)
        - "hello" + " world" (string concatenation, evaluates to "hello world")
        - len("python") (function call, evaluates to 6)
        - True or False (boolean operation, evaluates to True)
2. **Statement**
    - **What it is:** A complete instruction that the Python interpreter can execute. A statement performs some **action** or creates some effect. An expression can be part of a statement, but a statement does not necessarily evaluate to a value on its own (though some do).
    - **Execution Point of View:** Python executes statements sequentially to perform tasks.
    - **Examples:**
        - x = 10 (Assignment statement: assigns the value of the expression 10 to variable x)

- print("Hello") (Function call as a statement: performs the action of printing)
- if x > 5: (Conditional statement: controls flow)
- for item in my_list: (Loop statement: controls iteration)
- import os (Import statement: brings in a module)
- pass (Null operation statement: does nothing)
- return result (Return statement: exits a function and optionally returns a value)

**3. Line of Code**

- **What it is:** This refers to a single physical line of text in your Python script.
- **Execution Point of View:** While often containing a single statement, a "line of code" is a physical layout concept.
    - It can contain **multiple statements** separated by semicolons (though generally discouraged for readability): x = 10; y = 20; print(x+y)
    - A single statement can **span multiple lines** using parentheses, brackets, braces, or explicit backslashes for continuation:
    - A line can also be empty or contain only comments.

```
Python

my_long_list = [1, 2, 3,
                4, 5, 6]
```

**4. Code Block**

- **What it is:** A group of one or more statements that are treated as a single logical unit. In Python, code blocks are defined by **indentation**.
- **Execution Point of View:** These blocks are executed conditionally, iteratively, or as part of a function/class definition. Python's interpreter uses indentation to understand the structure and hierarchy of your code.

- **Examples:**
    - The statements under an if, elif, else clause.
    - The statements inside a for or while loop.
    - The statements inside a function definition (def).
    - The statements inside a class definition (class).
    - The statements inside a try...except...finally block.

```python
if temperature > 25:
    # This is a code block
    print("It's hot!")
    status = "high"
else:
    # This is another code block
    print("It's pleasant.")
    status = "normal"
```

### 5. Function

- **What it is:** A named, reusable block of code that performs a specific task. Functions encapsulate code to promote modularity, reusability, and readability.
- **Execution Point of View:** A function's code block is only executed when the function is *called*. When a function is called, a new local scope is created for its variables.
- **Example:**

```python
def greet(name): # Function definition
    # This indented part is the function's code block
    message = f"Hello, {name}!" # Local variable
    print(message)

greet("Alice") # Function call
```

6. **Class**

- **What it is:** A blueprint for creating objects (instances). It defines a set of attributes (data) and methods (functions) that the objects will have.
- **Execution Point of View:** When a class is defined, its code block (containing method definitions and class variables) is executed. However, the methods themselves are only executed when called on an instance of the class. Creating an object from a class (instantiation) also involves execution (e.g., __init__ method).
- **Example:**

Python

```
class Dog: # Class definition
    def __init__(self, name): # Method definition within the class
        self.name = name
    def bark(self):
        print(f"{self.name} says Woof!")

my_dog = Dog("Buddy") # Instantiation (creates an object)
my_dog.bark() # Method call on the object
```

7. **Module**

- **What it is:** A single .py file containing Python code (functions, classes, variables, statements).
- **Execution Point of View:** When a module is imported for the first time, all the top-level statements and definitions within it are executed from top to bottom. This includes defining functions, classes, and assigning global variables.

- **Example:**
    - my_module.py:

```python
# This is executed when 'my_module' is imported
PI = 3.14159

def calculate_area(radius):
    return PI * radius * radius

print("my_module is being imported!") # This print statement runs on import
```

    - main_script.py:

```python
import my_module # This executes my_module.py

area = my_module.calculate_area(5)
print(f"Area: {area}")
```

## 8. Package

- **What it is:** A collection of related modules organized in a directory hierarchy. A directory becomes a Python package if it contains an __init__.py file (which can be empty).
- **Execution Point of View:** When a package or a submodule within it is imported, the __init__.py files within the package (and its subpackages, if applicable) are executed.
- **Example:**
    - my_package/
        - __init__.py
        - math_ops.py (module)
        - string_utils.py (module)
    - import my_package.math_ops (This would execute my_package/__init__.py and then my_package/math_ops.py)

**9. Program / Script**

- **What it is:** The complete set of Python files (one main script, and any modules/packages it imports) that are executed from start to finish.
- **Execution Point of View:** The entire process of the Python interpreter loading, parsing, and running the code, starting from the entry point (the main script).

Understanding this hierarchy helps in debugging, organizing code, and comprehending how Python manages execution flow and namespaces.