1-which languages that have GC and not?

Languages with Garbage Collection:

1. **Java** - Uses a sophisticated garbage collector with multiple algorithms (e.g., G1, ZGC).
2. **C#** - Part of the .NET framework, which includes an advanced garbage collector.
3. **Python** - Uses reference counting and a cyclic garbage collector.
4. **Ruby** - Has a mark-and-sweep garbage collector.
5. **JavaScript** - Garbage collection is managed by the JavaScript engine (V8, SpiderMonkey, etc.).
6. **Go** - Designed with garbage collection to help manage memory in concurrent programs.
7. **Swift** - Uses Automatic Reference Counting (ARC), which is similar to garbage collection.
8. **Scala** - Runs on the JVM and benefits from Java's garbage collector.
9. **Haskell** - Uses garbage collection to manage memory in its functional paradigm.
10. **Clojure** - Runs on the JVM, hence uses Java's garbage collector.

Languages without Garbage Collection (Manual Memory Management):

1. **C** - Requires explicit memory management using `malloc` and `free`.
2. **C++** - Uses `new` and `delete` for memory management, though smart pointers can help.
3. **Rust** - Uses a system of ownership with rules that the compiler checks at compile time.
4. **Assembly** - Directly manages memory without any built-in garbage collection.
5. **Fortran** - Typically uses manual memory management, though some modern versions might offer limited garbage collection.
6. **Pascal** - Manual memory management using `new` and `dispose`.

Languages with Optional or Mixed Garbage Collection:

1. **Objective-C** - Uses Automatic Reference Counting (ARC) in some environments, manual memory management in others.
2. **D** - Has optional garbage collection; can use manual memory management if desired.

Garbage collection can simplify memory management and reduce memory leaks but can also introduce overhead. Conversely, manual memory management provides more control but can be error-prone.

2- what 13 principle for clean code?

- **Meaningful Names**:
    - Use descriptive and unambiguous names.
    - Make clear distinctions between similar items.
    - Use pronounceable and searchable names.

- **Functions Should Be Small**:
  - Keep functions short, ideally around 20 lines or less.
  - Each function should do one thing and do it well (Single Responsibility Principle).
- **Functions Should Have Few Arguments**:
  - Aim for no more than three arguments in a function.
  - Use object parameters if more arguments are needed.
- **Avoid Side Effects**:
  - Functions should not change the state of objects outside their scope.
  - Functions should not perform unexpected actions.
- **Command Query Separation**:
  - Functions should either do something (commands) or answer something (queries), but not both.
- **Prefer Exceptions to Returning Error Codes**:
  - Use exceptions for error handling, not error codes.
  - This makes the code cleaner and more readable.
- **DRY Principle (Don't Repeat Yourself)**:
  - Avoid duplicating code.
  - Abstract common functionality into functions or classes.
- **Encapsulation**:
  - Keep data and behavior related to that data together.
  - Use private fields and methods to hide implementation details.
- **Separation of Concerns**:
  - Each class or function should have a single responsibility.
  - Different concerns should be handled in separate parts of the code.
- **Use Polymorphism to Replace If/Else or Switch/Case Statements**:
  - Replace complex conditional logic with polymorphic methods or strategy patterns.
- **Class and Method Naming**:
  - Class names should be nouns or noun phrases.
  - Method names should be verbs or verb phrases.

- **Clean Code is Simple and Direct**:
    - Avoid unnecessary complexity.
    - Code should be straightforward, even if that means writing more lines.

- **Code Should Be Easy to Read and Understand**:
    - Use whitespace to separate different sections of code.
    - Comment only what is necessary, ensuring the code itself is self-explanatory.
    - Use consistent naming conventions and code formatting.

3-How to implement do while in python?

```
while True:
    # Code to execute
    if not condition:
        break
```

4-The difference between For and While loops in python?

for loop:

- **Iterates Over a Sequence**: Typically used for iterating over sequences like lists, strings, or sets.
- **Ease of Use**: Clear and straightforward when iterating over known-size collections.
- **Example**:

```
for i in range(5):
       print(i)
```

while loop:

- **Condition-Based Iteration**: Used when iterating until a specific condition is met.
- **Flexibility**: Offers more flexibility but requires careful handling of the condition to avoid infinite loops.
- **Example**:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

5-Equivelant of (pass) in java and c++?

for loop:

- **Iterates Over a Sequence**: Typically used for iterating over sequences like lists, strings, or sets.
- **Ease of Use**: Clear and straightforward when iterating over known-size collections.
- **Example**:

```
for i in range(5):
        print(i)
```

`while` loop:

- **Condition-Based Iteration**: Used when iterating until a specific condition is met.
- **Flexibility**: Offers more flexibility but requires careful handling of the condition to avoid infinite loops.
- **Example**:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

6-Popular tracing tools in python?

`for` loop:

- **Iterates Over a Sequence**: Typically used for iterating over sequences like lists, strings, or sets.
- **Ease of Use**: Clear and straightforward when iterating over known-size collections.
- **Example**:

```
for i in range(5):
        print(i)
```

`while` loop:

- **Condition-Based Iteration**: Used when iterating until a specific condition is met.
- **Flexibility**: Offers more flexibility but requires careful handling of the condition to avoid infinite loops.
- **Example**:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

7-How write in the middle of file?

If you want to overwrite content in the middle without inserting new content, you can use(`seek)` :

```
with open('file.txt', 'r+') as file:
    # Seek to the middle of the file
    file.seek(len(file.read()) // 2)
    # Write text in the middle of the file
    file.write('This is new text')
```