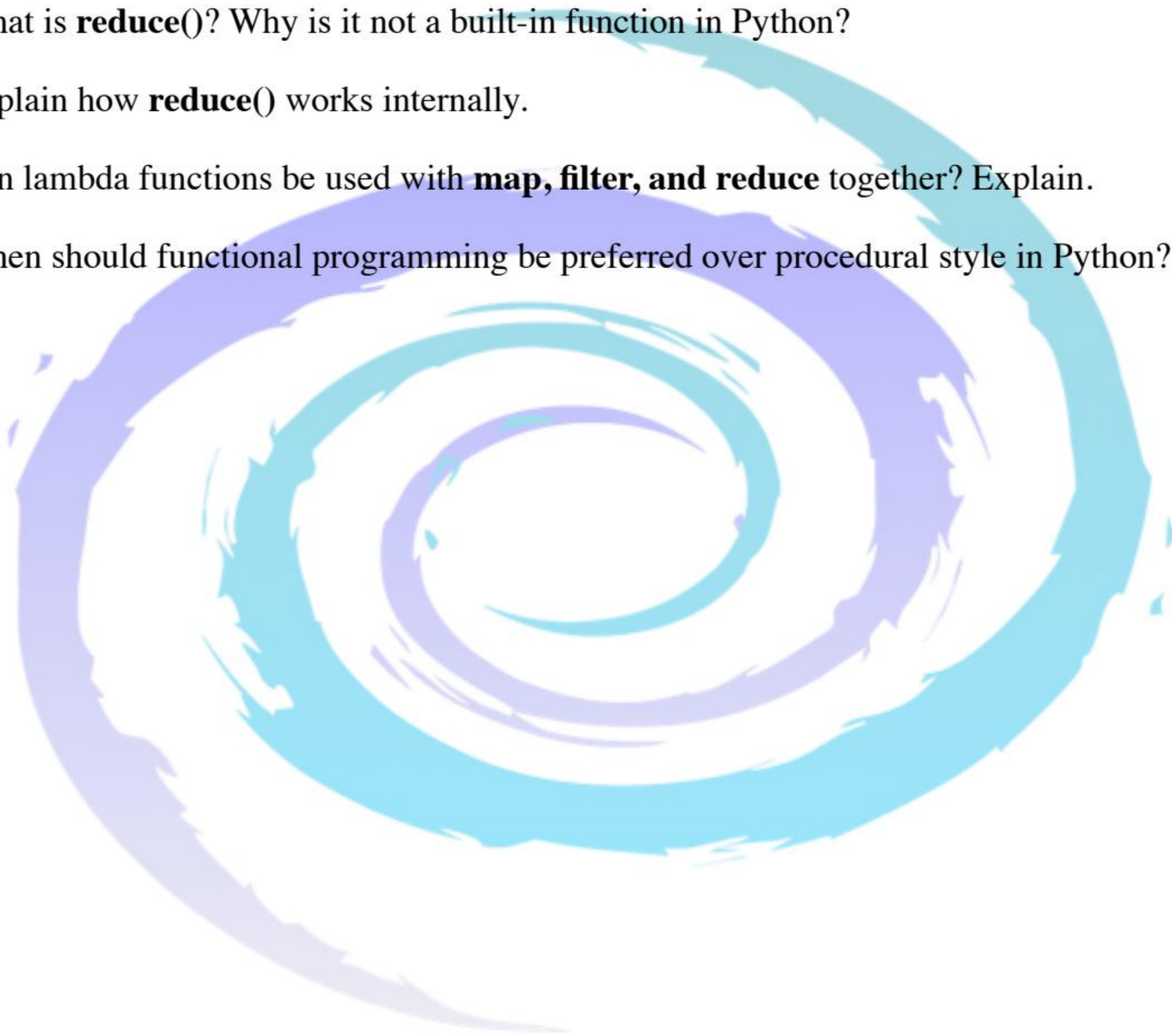
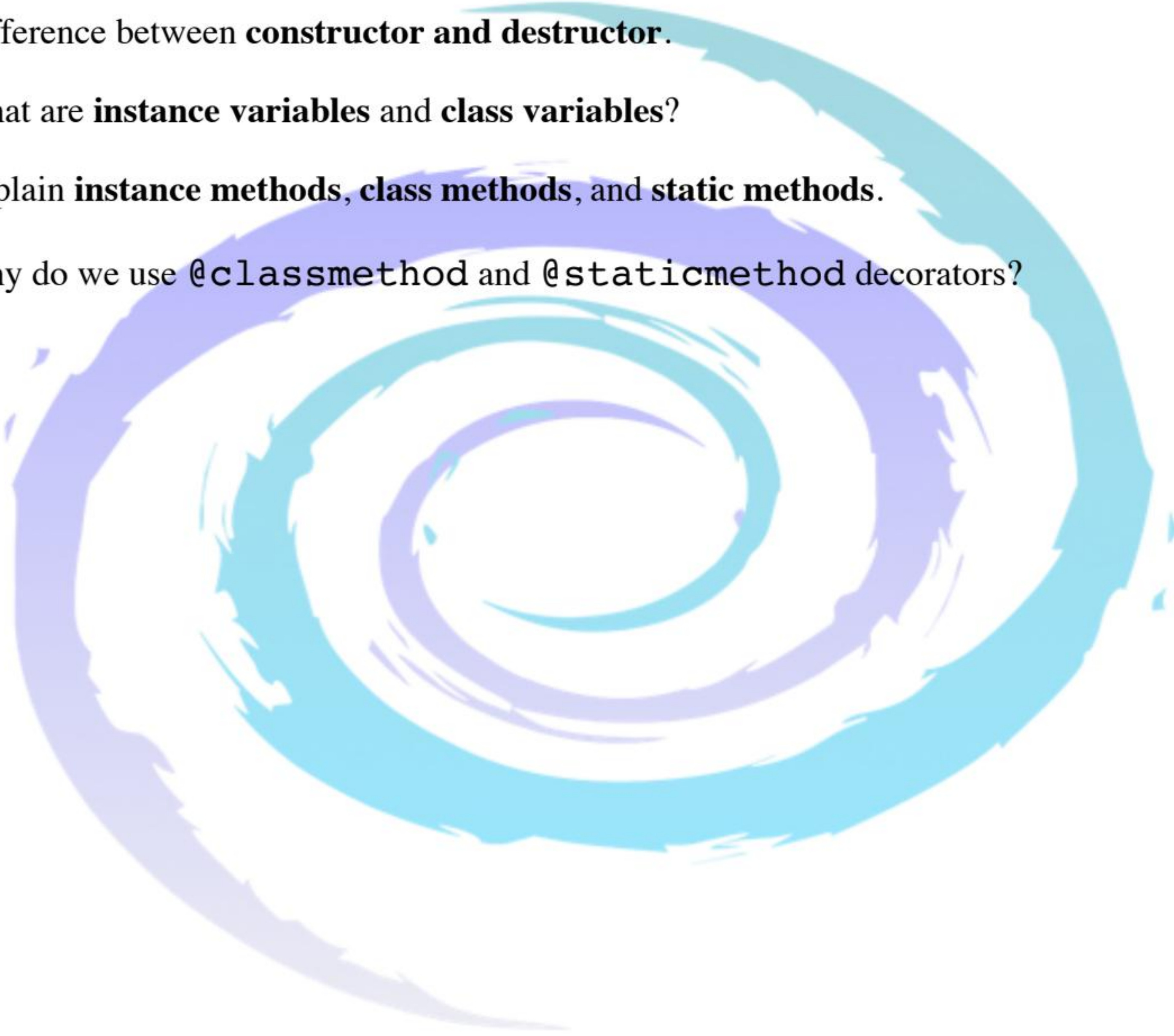


## Python Programming

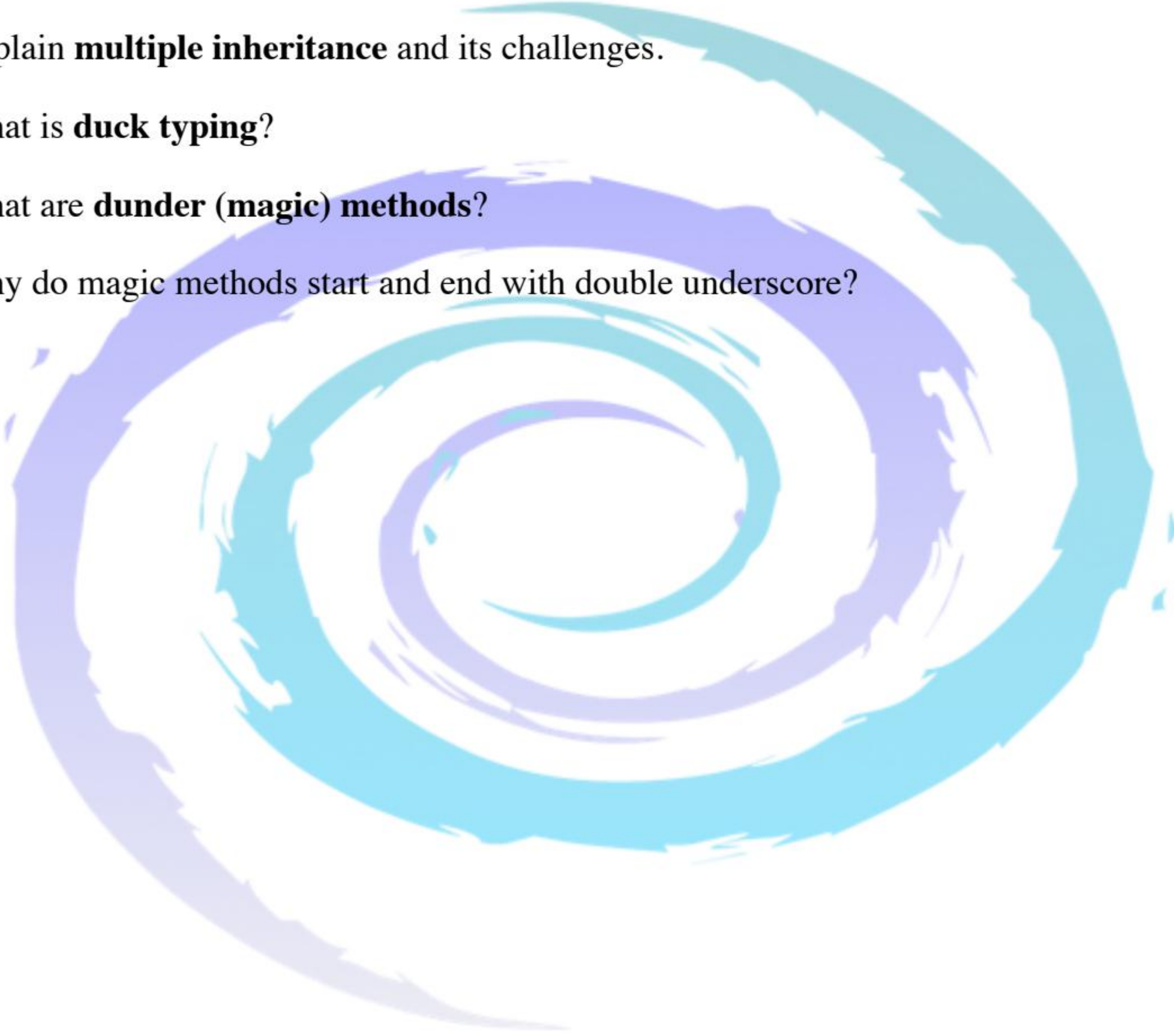
1. What is a **lambda function** in Python? How is it different from a normal function?
2. What are the **limitations of lambda functions**?
3. Explain the working of the **map()** function with an example.
4. How does **map()** differ from using a **for** loop?
5. What is the **filter()** function and when should it be used?
6. Difference between **map()** and **filter()**.
7. What is **reduce()**? Why is it not a built-in function in Python?
8. Explain how **reduce()** works internally.
9. Can lambda functions be used with **map, filter, and reduce** together? Explain.
10. When should functional programming be preferred over procedural style in Python?



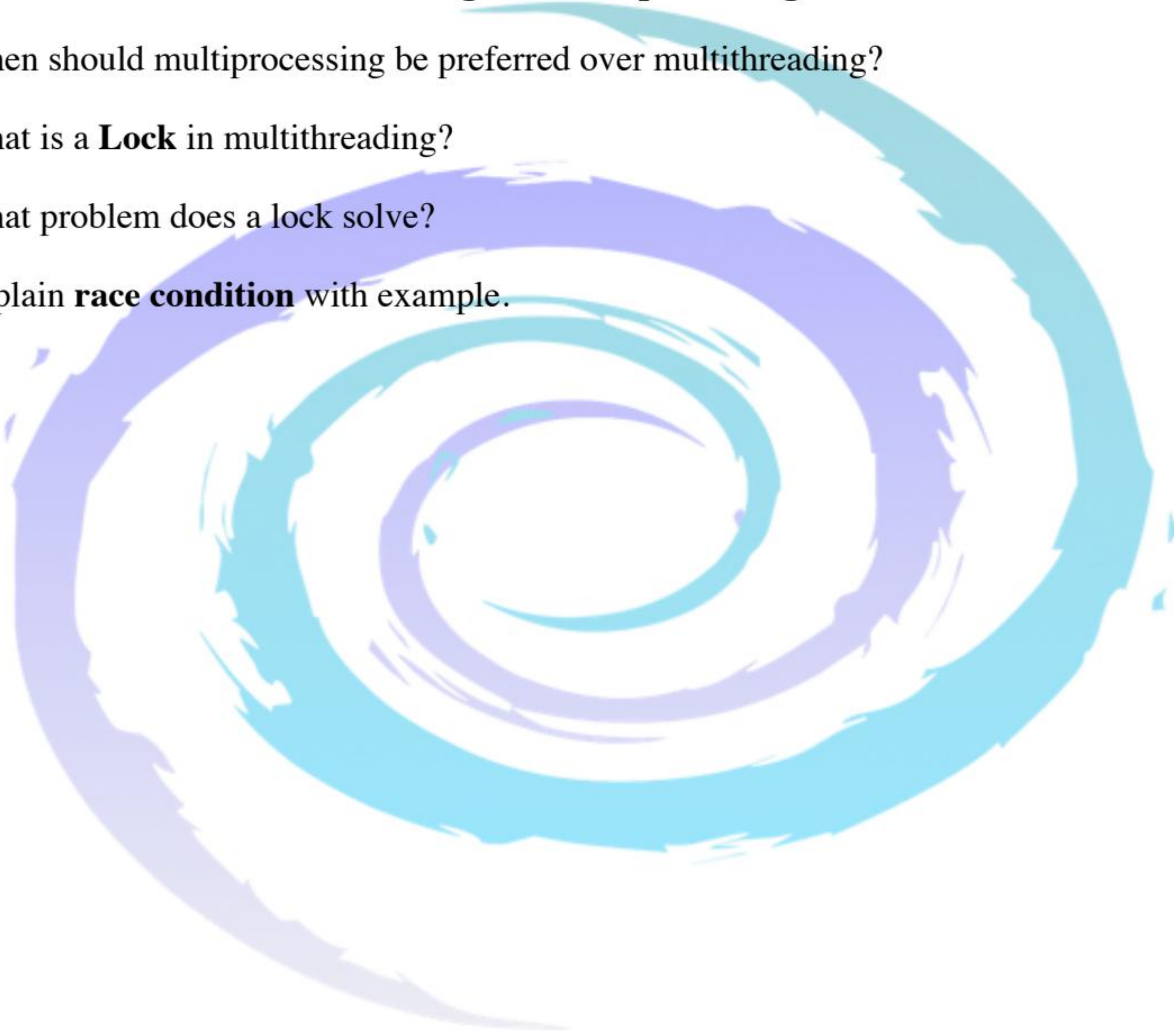
## Python Programming

1. What is **Object-Oriented Programming**?
  2. What are the **four core characteristics of OOP**?
  3. Explain **class and object** with real-world analogy.
  4. What is a **constructor** in Python? Why is `__init__()` used?
  5. Is constructor mandatory in a class? Explain.
  6. What is a **destructor**? When is `__del__()` called?
  7. Difference between **constructor and destructor**.
  8. What are **instance variables** and **class variables**?
  9. Explain **instance methods, class methods, and static methods**.
  10. Why do we use `@classmethod` and `@staticmethod` decorators?
- 

## Python Programming

1. What is **encapsulation** and how is it implemented in Python?
  2. Explain **polymorphism** in Python.
  3. What is **method overriding**?
  4. What is **super()** and why is it used?
  5. What is **inheritance** in Python?
  6. What are the **types of inheritance** supported by Python?
  7. Explain **multiple inheritance** and its challenges.
  8. What is **duck typing**?
  9. What are **dunder (magic) methods**?
  10. Why do magic methods start and end with double underscore?
- 

## Python Programming

1. What is the difference between `__add__()` and `__iadd__()`?
  2. How does operator overloading work using magic methods?
  3. Explain `__len__()`, `__eq__()`, and `__lt__()`.
  4. What is **multithreading** in Python?
  5. What is the **Global Interpreter Lock (GIL)**?
  6. Difference between **multithreading** and **multiprocessing**.
  7. When should multiprocessing be preferred over multithreading?
  8. What is a **Lock** in multithreading?
  9. What problem does a lock solve?
  10. Explain **race condition** with example.
- 

## Python Programming

1. What is a **Process Pool**?
  2. What is **scheduling in Python**?
  3. Difference between `time.sleep()` and scheduler.
  4. What are **command line arguments**?
  5. How does `sys.argv` work?
  6. Difference between `input()` and **command line arguments**.
  7. What is **exception handling**?
  8. Difference between **syntax error** and **runtime error**.
  9. Explain `try`, `except`, `else`, and `finally`.
  10. What is **the use of PIP utility**?
- 