Python's memory management.
• Uses garbage collection (removes unused objects).
• Uses reference counting (sys.getrefcount(obj)).
• Large objects go to the heap.

Python's control flow statements
• Conditional statements: if, elif, else
• Loops: for, while
• Loop control: break, continue, pass

• Shallow copy: Copies object but not nested objects (copy.copy()).
• Deep copy: Copies object and nested objects (copy.deepcopy()).

• Mutable: Can be modified (list, dict, set).
• Immutable: Cannot be modified (int, str, tuple).

Difference between classmethod, staticmethod, and instance methods
• Instance Method: Works with object instance (self).
• Class Method: Works with class (cls, @classmethod).
• Static Method: No self or cls, acts like a normal function.

• Static method: No self, acts like a normal function.
• Class method: Uses cls and works on the class level.

Python's Global Interpreter Lock (GIL)
• GIL prevents multiple threads from executing Python bytecode simultaneously.
• This limits Python's true multithreading capabilities.
Solution: Use multiprocessing instead of multithreading for CPU-bound tasks.

map(), filter(), and reduce()
• map(func, iter): Applies func to all elements.
• filter(func, iter): Filters elements where func returns True.
• reduce(func, iter): Applies cumulative function (import from functools).

Python's assert statement - Used for debugging and unit testing.

Python frozen set - Immutable version of a set.

Python decorator
• A decorator modifies the behavior of a function without changing its code.

Metaprogramming is writing code that modifies code at runtime (e.g., metaclasses).

Monkey patching is modifying a class at runtime.

Duck typing in Python - If an object behaves like a type, it is that type.

Characteristics of a Python set
• Unordered collection
• No duplicate elements
• Mutable

Uses yield instead of return to create iterators lazily.

Types of Inheritance in Python

1. **Single Inheritance – One child class inherits from one parent class.**
2. **Multiple Inheritance – A child class inherits from multiple parent classes.**
3. **Multilevel Inheritance – A class inherits from another class, which in turn inherits from another class.**
4. **Hierarchical Inheritance – Multiple child classes inherit from a single parent class.**
5. **Hybrid Inheritance – A combination of multiple types of inheritance.**

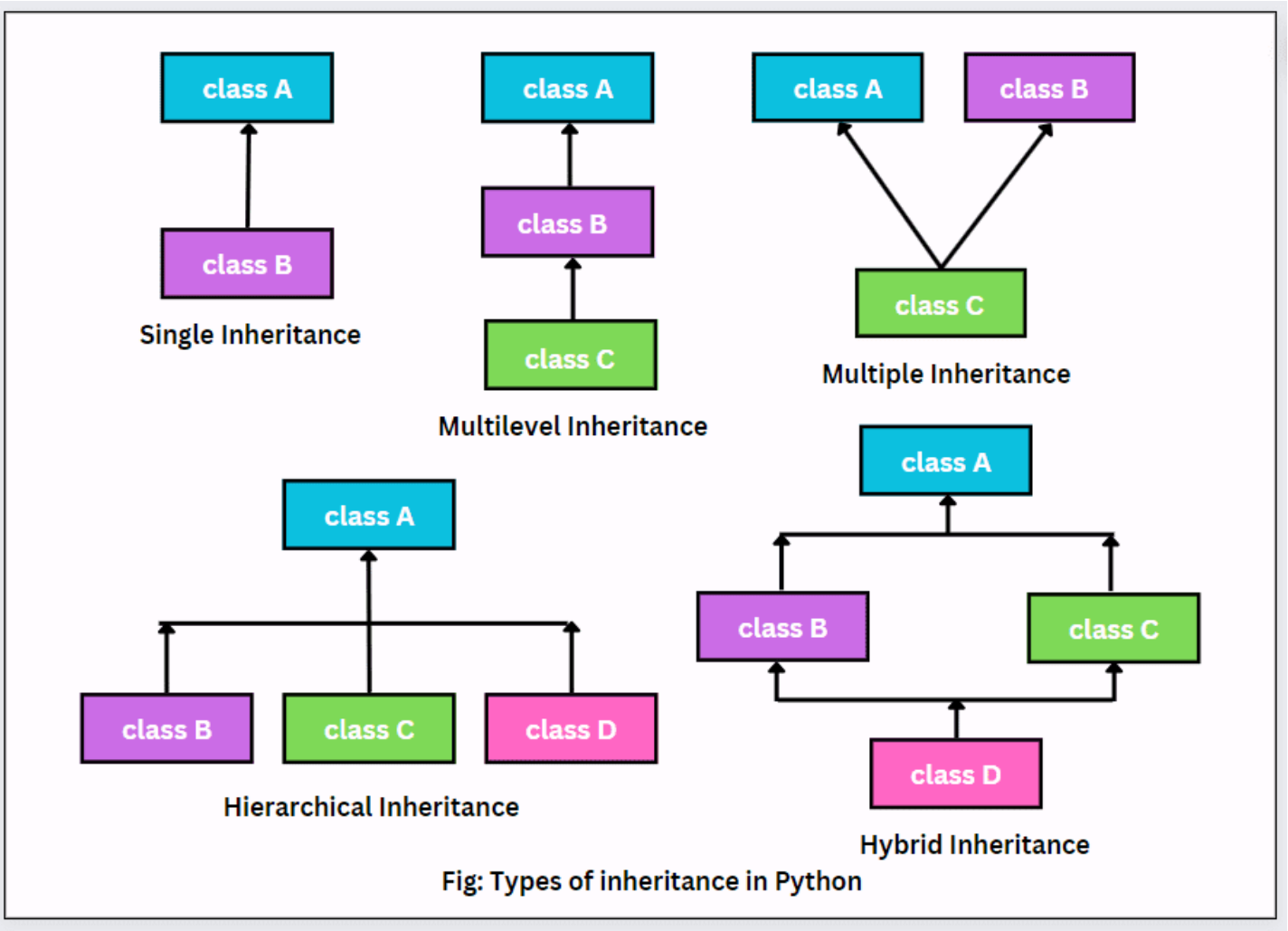

Fig: Types of inheritance in Python

Method Overriding in Inheritance:
If a child class has the same method as the parent class, the child class method overrides the parent's method.

The super() function allows a child class to access methods of its parent class.

Method resolution order (MRO):
• MRO defines the order in which base classes are searched for a method or attribute.

In [3]:

```
add = lambda x,y: x+y
print(add(2,3))
```

5

In [96]:

```
import copy
list1 = [[1,2],[3,4]]
shallow = copy.copy(list1)
deep = copy.deepcopy(list1)

list1[0][0] = 99

print(shallow)
```

```
print(deep)
```

```
[[99, 2], [3, 4]]
[[1, 2], [3, 4]]
```

In [6]:

```
try:
    x=1/0
except ZeroDivisionError:
    print("Can't divide by 0")
finally:
    print("Execution completed")
```

```
Can't divide by 0
Execution completed
```

In [14]:

```
str = "Arunoth Symen"
words = str.split()
print(words)
print(" ".join(words))
```

```
['Arunoth', 'Symen']
Arunoth Symen
```

In [21]:

```
n = int(input())
for i in range(1, n+1):
    for j in range(1, i+1):
        print(j, end=" ")
    print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

In [120]:

```
n = int(input("Enter the number of elements: "))
arr = []

print("Enter the elements:")
for _ in range(n):
    num = int(input())
    arr.append(num)

# Find the largest element manually
largest = arr[0]
for i in range(1, n):
    if arr[i] > largest:
        largest = arr[i]

print(arr)
print("Largest element is:", largest)
```

```
Enter the elements:
[3, 5, 6, 8, 1]
Largest element is: 8
```

In [27]:

```
str = "Symen"
rev_str = str[::-1]
print(rev_str)
```

```
nemyS
```

In [50]:

```
def printArray(arr, n):
    for i in range(n):
        print(arr[i], end="")
    print()

def reverseArray(arr, n):
    ans = [0] * n
    for i in range(n - 1, -1, -1):
        ans[n - i - 1] = arr[i]
    printArray(ans, n)

if __name__ == "__main__":
    arr = ["n","e","m","y","S"]
    n = len(arr)
    reverseArray(arr, n)
```

```
Symen
```

In [54]:

```
str = "Symen"
rev_s = ""

for i in range(len(str) -1, -1, -1):
    rev_s = rev_s + str[i]
print(rev_s)
```

```
nemyS
```

In [57]:

```
def is_palindrome(s):
    n = len(s)
    for i in range(n//2):
        if s[i] != s[n -i -1]:
            return False
    return True

s = input("Enter a string: ")

if is_palindrome(s):
    print("True")
else:
    print("False")
```

```
True
```

In [63]:

```
def reverse_word(rw):
    word = rw.strip().split()
    reverse_word = ' '.join(word[::-1])
    return reverse_word

rw = input("Enter a sentence: ")

print(reverse_word(rw))
```

```
Symen I'm Hii!
```

In [ ]:

```
#with open("file.txt","w") as f:
#    f.write("Hello Symen")

#with open("file.txt","r") as f:
#    print(f.read())
```

```
Hello Symen
```

In [79]:

```
names = ["Arunoth","Symen"]
for index, name in enumerate(names):
    print(index,name)
```

```
0 Arunoth
1 Symen
```

In [83]:

```
a = [1,2,3,4]
b = ["Antony","Hanna","Allwin","Symen"]

print(list(zip(a,b)))
```

```
[(1, 'Antony'), (2, 'Hanna'), (3, 'Allwin'), (4, 'Symen')]
```

In [90]:

```
from functools import reduce
nums = [1,2,3,4]

print(list(map(lambda x:x**2,nums)))
print(list(filter(lambda x:x%2==0,nums)))
print(reduce(lambda x,y:x+y,nums))
```

```
[1, 4, 9, 16]
[2, 4]
10
```

In [100]:

```
list = [1,2,3,4]
tup = tuple(list)

print(tup)
```

```
(1, 2, 3, 4)
```

In [105]:

```
tuple = ([1,2],[3,4])
tuple[1].append(5)
print(tuple)
```

```
([1, 2], [3, 4, 5])
```

In [106]:

```
tup = (10,20,30)
a,b,c = tup
print(a)
```

```
10
```

In [108]:

```
a = {1, 2}
b = {1, 2, 3, 4}
print(a.issubset(b))
print(b.issuperset(a))
```

```
True
True
```

In [109]:

```
d1 = {"a": 1, "b": 2}
d2 = {"c": 3, "d": 4}
merged = {**d1, **d2}
print(merged)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

In [117]:

```python
class Car:
    # Constructor method to initialize object properties
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    # Method to display car details
    def show_details(self):
        print(f"Car Brand: {self.brand}")
        print(f"Car Model: {self.model}")
        print(f"Manufacturing Year: {self.year}")

# Creating objects of class Car
car1 = Car("Toyota", "Camry", 2022)
car2 = Car("Honda", "Civic", 2021)

# Calling methods using objects
car1.show_details()
print()  # Blank line for better readability
car2.show_details()
```

```
Car Brand: Toyota
Car Model: Camry
Manufacturing Year: 2022

Car Brand: Honda
Car Model: Civic
Manufacturing Year: 2021
```