Dunder Methods Cheat Sheet

Reference: https://docs.python.org/3/reference/datamodel.html#special-method-names

Prepared By: Shehab Abdel-Salam

1. `__init__(self, ...)` - The constructor method.

Purpose: Called when an object is created. Used to initialise the object's attributes.

```
class Person:
    def __init__(self, name):
        self.name = name
```

2. `__str__(self)` - String representation for end-users.

Purpose: Defines what is shown when `print()` or `str()` is called on an object. Returns a readable string version of the object.

```
class Person:
    def __str__(self):
        return f"Person: {self.name}"
```

3. `__repr__(self)` - Developer-friendly representation.

Purpose: Used for debugging and development. Returns a more detailed string often used in logs or console.

```
class Person:
    def __repr__(self):
```

```
        return f"Person(name={self.name})"
```

4. `__len__(self)` - Length of an object.

Purpose: Allows the use of `len()` to return the length of an object.

```
class Group:
    def __len__(self):
        return len(self.members)
```

5. `__getitem__(self, key)` - Accessing elements via index/key.

Purpose: Allows objects to support indexing (e.g., `obj[key]`).

```
class MyList:
    def __getitem__(self, index):
        return self.data[index]
```

6. `__setitem__(self, key, value)` - Set an element at an index/key.

Purpose: Allows setting values in objects using indexing (e.g., `obj[key] = value`).

```
class MyList:
    def __setitem__(self, index, value):
        self.data[index] = value
```

7. `__delitem__(self, key)` - Delete an element at an index/key.

Purpose: Allows removing values from an object using indexing (e.g., `del obj[key]`).

```
class MyList:
    def __delitem__(self, index):
        del self.data[index]
```

8. `__call__(self, *args, **kwargs)` - Make an object callable.

Purpose: Allows instances of the class to be called like a function.

```
class Adder:
    def __call__(self, a, b):
        return a + b
```

9. `__iter__(self)` - Returns an iterator.

Purpose: Allows the object to be iterable in loops (e.g., `for item in obj`).

```
class MyList:
    def __iter__(self):
        return iter(self.data)
```

10. `__next__(self)` - Returns the next element in iteration.

Purpose: Used in conjunction with `__iter__` to move through the elements of an object.

```
class MyIterator:
    def __next__(self):
        # Logic to return the next item
```

11. `__eq__(self, other)` - Equality comparison.

Purpose: Defines the behaviour for `==`. Allows custom comparison of two objects.

```
class Person:
    def __eq__(self, other):
        return self.name == other.name
```

12. `__lt__(self, other)` - Less than comparison.

Purpose: Defines the behaviour for `<` (useful for sorting).

```
class Person:
    def __lt__(self, other):
        return self.age < other.age
```

13. `__add__(self, other)` - Addition operation.

Purpose: Defines the behaviour for `+` when adding two objects.

```
class Vector:
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y
```