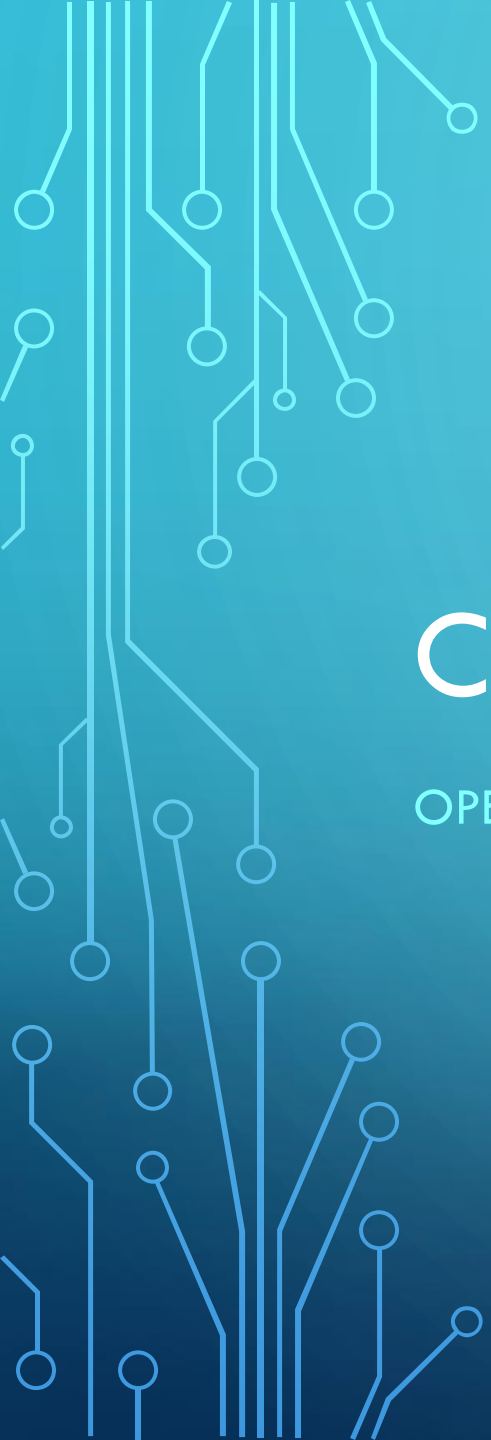

CLASS INTERNALS

OPERATOR OVERLOADING

Operator Overloading

- Python operators work for built-in classes, but the same operator behaves differently with different types. For example, the + operator will, perform arithmetic addition on two numbers, merge two lists and concatenate two strings.
- Operator overloading allows same operator to have different meaning based on the context.
- User defined classes can overload operators too. To overload the + sign, for example, implement the __add__() function in the class.

Overloading Example

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def __str__(self):
        return str(self.x) + str(self.y)
    def __add__(self,rhs):
        x = self.x + rhs.x
        y = self.y + rhs.y
        return Point(x,y)
```

Usage:

```
p1 = Point(100,100)
p2 = Point(200,200)
pAdd = p1 + p2
```

Comparison Overloading

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

Mathmatical Overloading

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

Iterating

- Iterators are implemented in most computer programming languages including Python. An Iterator is an object that can be iterated. An iteration object will return data, one element at a time from a collection of objects (like a list).

- An iterator object must implement two special methods, __iter__() and __next__(), called the iterator protocol.

Iterating Example

```
class PowTwo:
    def __init__(self, max = 0):
        self.max = max
    def __iter__(self):
        self.n = 0
        return self
    def __next__(self):
        if self.n <= self.max:
            result = 2 ** self.n
            self.n += 1
            return result
        else:
            raise StopIteration
```

Usage:

```
p = PowTwo(4)
i = iter(p)
for j in range(0,4)
    print(next(i))
```