Object Oriented Programming

Python 101 - Week 5

Mehmet Arif Demirtaş
marifdemirtas.github.io

Recap: Data Types

Data Type	Literals		
`int`	`1, -4, 0, 10000`	Numeric types Supports addition, multiplication	Scalars
`float`	`1.5, -3.2, 9.25, 15.0`		
`bool`	`True, False`	Supports logical comparisons (<, >)	
`NoneType`	`None`	Placeholder for nothing	
`str`	`'a', "abcd", 'hello world!'`	Text	Collections
`tuple`	`(1,), ('a', 3, 8), ((1, 2), 3)`	Items of different types	
`range`	`range(start, stop, step)`	Numerical ranges	
`list`	`[1], ['a', 3, 8], [[1, 2], 3]`	Items of different types, can be mutated	
`function`	…	Functions are data types too!	

Recap: Control Flow Structures

Conditional (if-else)	while (loop)	for (loop)

`test1 = (5<3) and (10>2)`
`test2 = (1!=1) or (not False)`
 	
`if test1:`
`  print("Test is true")`
`elif test2:`
`  print("Test2 is true")`
`else:`
`  print("Both are false")`
	`i = 0`
	`N = 6`
	
	`while i < N:`
	`  print(i * 2 + 1)`
	`  i += 2`
		`for num in range(0, 6, 2):`
		`  print(num * 2 + 1)`

`Test2 is true`	**`1`** **`5`** **`9`**	

Recap: Functions

Function Definition	Function Call	Result

`def func1():`
`  print("This is func 1")`
	`x = 5`
	`func1()`
	`x += 1`
		`This is func 1`

`def func2(x):` `  print(x ** 3)`	`a = 4` `func2(7)`	`64`

`def func3(x, y):`
`  return x + 2 * y`
	`a, b = 3, 5`
	`c = func(a, b)`
	`print(c)`
		`13`

`from math import sqrt as sq`
`def func4(t1, t2, sqrt=True):`
`  res = t1**2 + t2**2`
`  if sqrt:`
`  res = sq(res)`
`  return res`
	`a, b = 3, 4`
	`c1 = func4(a, b)`
	`print(f"Default: {c1}")`
	`c2 = func4(a, b, sqrt=False)`
	`print(f"No sqrt: {c2}")`
		`Default: 5.0`
		`No sqrt: 13`

Recap: Structure of a data type

```
Object int:
    name = a_number # variable name
    data = 0x010 # address

    functions:
        __add__: # add (+)
            ... # statements that will sum up two numbers
        __mul__:  # multiply (*)
            ... # statements that will multiply two numbers
```

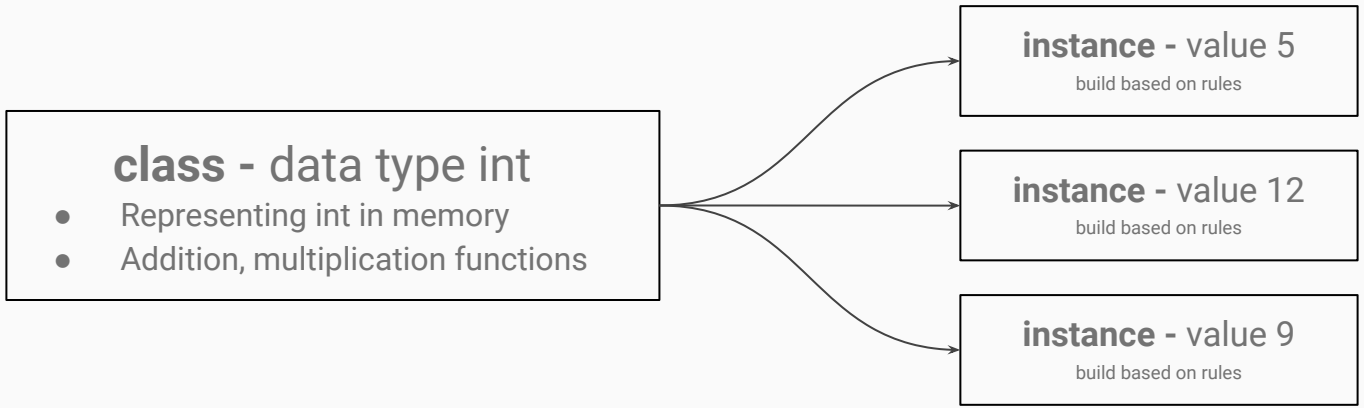
```
Object str:
    name = a_string # variable name
    data_start = 0x010 # start address
    data_end = 0x030 # end address

    functions:
        __add__: # add (+)
            ... # statements that will concatenate strings
        __mul__:  # multiply (*)
            ... # statements that will concatenate a string with itself N times
```

Structure of an object

- Internal data representation
- Functions for interacting with the object

An **class** is the *recipe/blueprint* for the **instance** of that type.



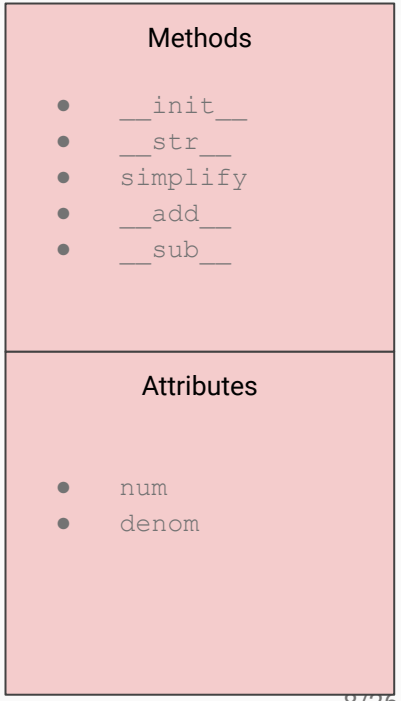
Why use objects?

- Collect a concept and everything you can do with that concept under a single structure
- Abstract away inner representation and interact with well-defined interface functions (methods on an object)

Example: Fraction

```
import math
class Fraction(object):
    ...
    def __sub__(self, other):
        other.num *= -1
        return self + other
```

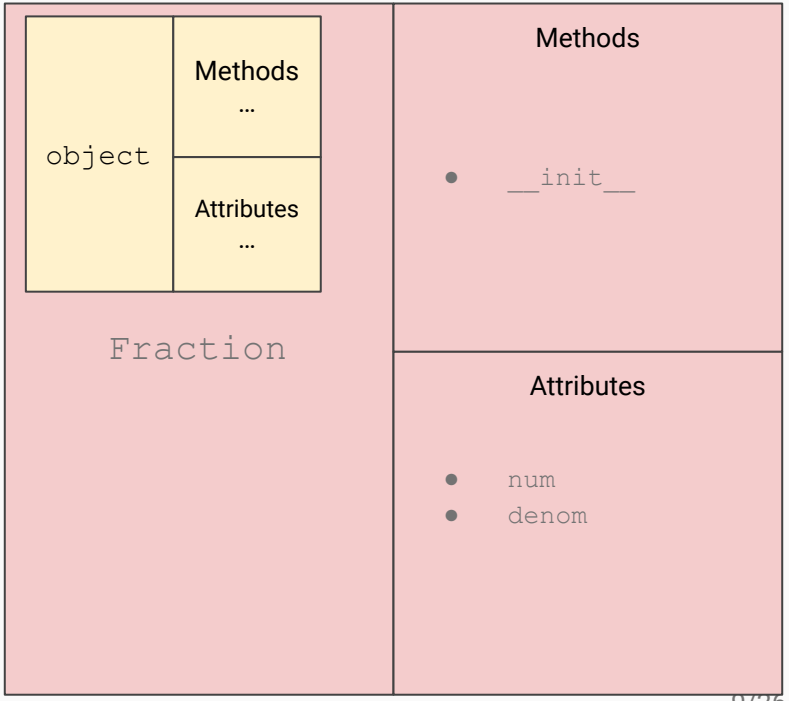
Define subtraction on Fractions by multiplying numerator by -1.



Example: Fraction

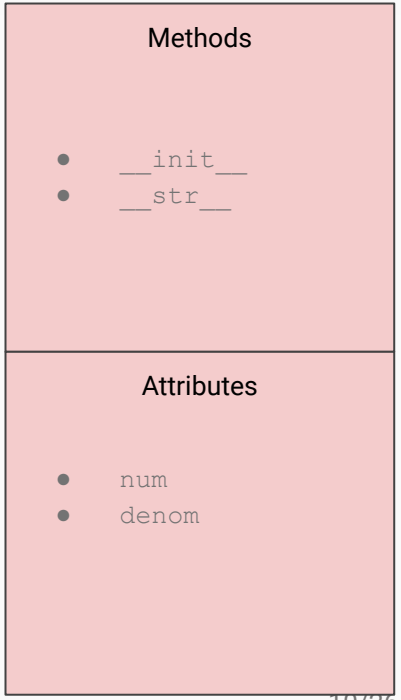
```
class Fraction(object):
    def __init__(self,
                 num,
                 denom):
        self.num = num
        self.denom = denom
```

`class`: Start an object recipe definition
`(object)`: **Inherit** from a parent
`__init__`: Define a **constructor** function
`self`: This is a class function, so it takes the object instance as first parameter.
`self.num`: The instance now has an **attribute** called num.



Example: Fraction - special methods

```
class Fraction(object):
    ...
    def __str__(self):
        return f"{self.num}/{self.denom}"
```



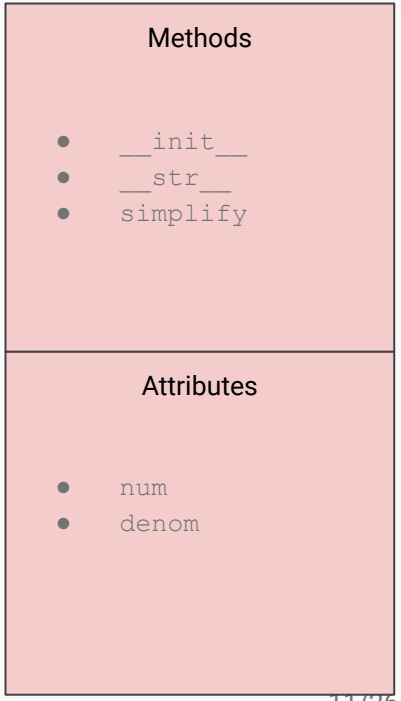
Define a special methods starting with double underscores that will be called by Python:

String representation of the instance

Example: Fraction

```
import math
class Fraction(object):
    ...
    def simplify(self):
        gcd = math.gcd(self.num, self.denom)
        self.num /= gcd
        self.denom /= gcd
```

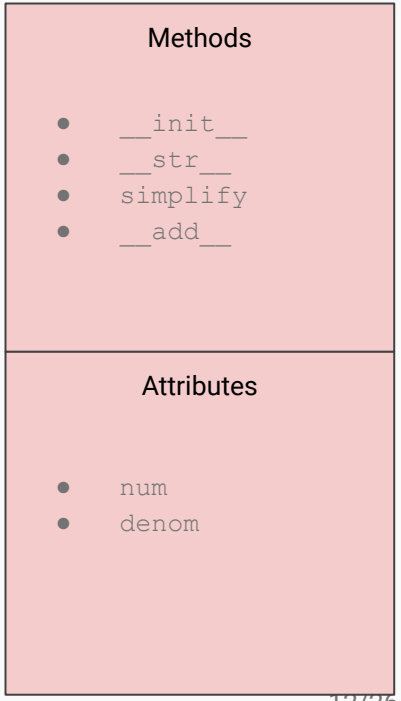
Simplify a fraction by importing **gcd** function and dividing numerator and denominator by gcd of two values.



Example: Fraction

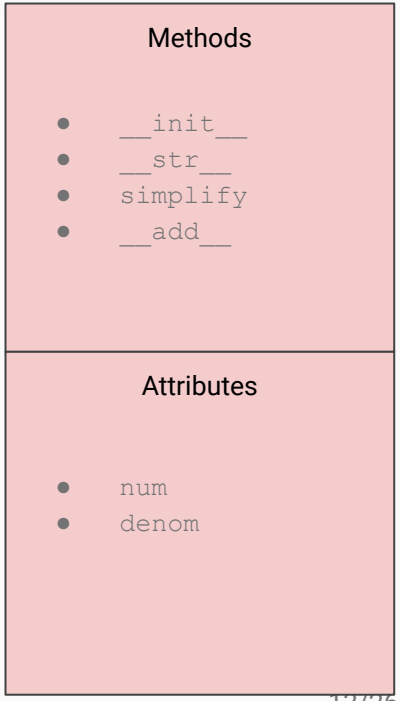
```
import math
class Fraction(object):
    ...
    def __add__(self, other):
        result = Fraction(
            num=(self.num * other.denom +
                 self.denom * other.num),
            denom=(self.denom * other.denom)
        )
        result.simplify()
        return result
```

Define addition on Fractions.



Example: Fraction

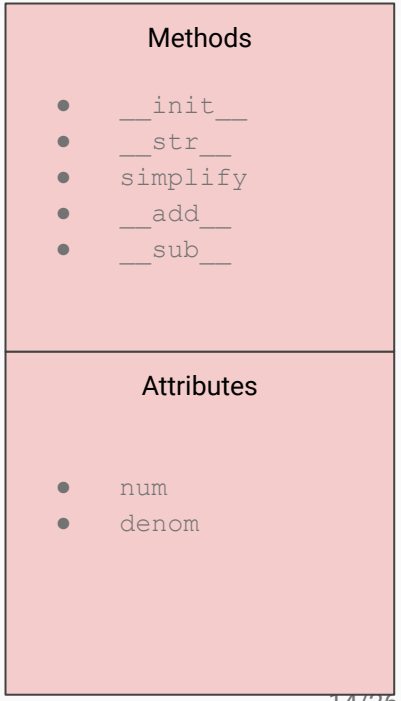
```
class Fraction(object):
    ...
    def __add__(self, other):
        if isinstance(other, int):
            other = Fraction(other, 1)
        result = Fraction(
            num=(self.num * other.denom +
                 self.denom * other.num),
            denom=(self.denom * other.denom)
        )
        result.simplify()
        return result
```



Example: Fraction

```
import math
class Fraction(object):
    ...
    def __mul__(self, other):
        TODO
```

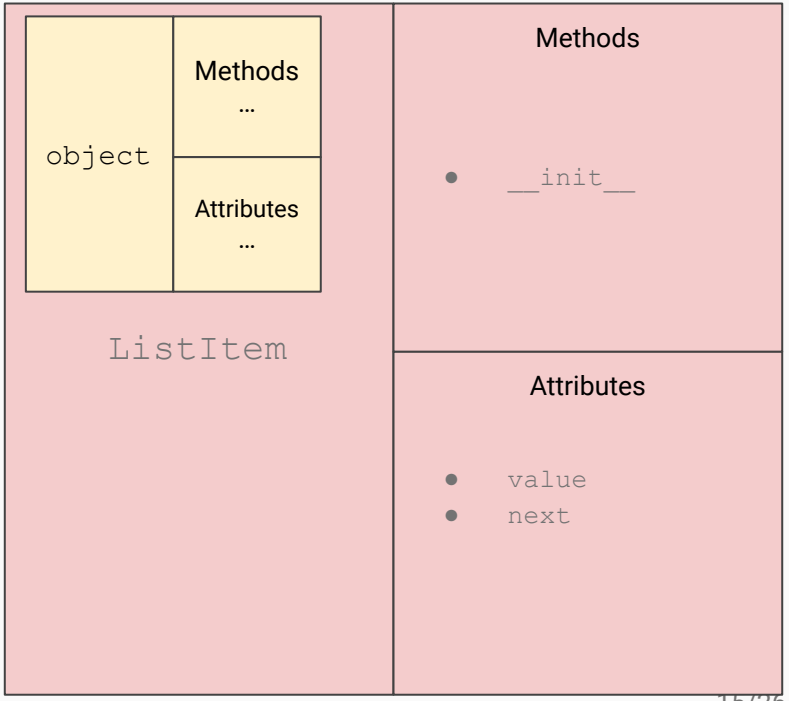
Define multiplication on fractions.



Example: Lists from scratch

```
class ListItem(object):
    def __init__(self, value, next=None):
        self.value = value
        self.next = next
```

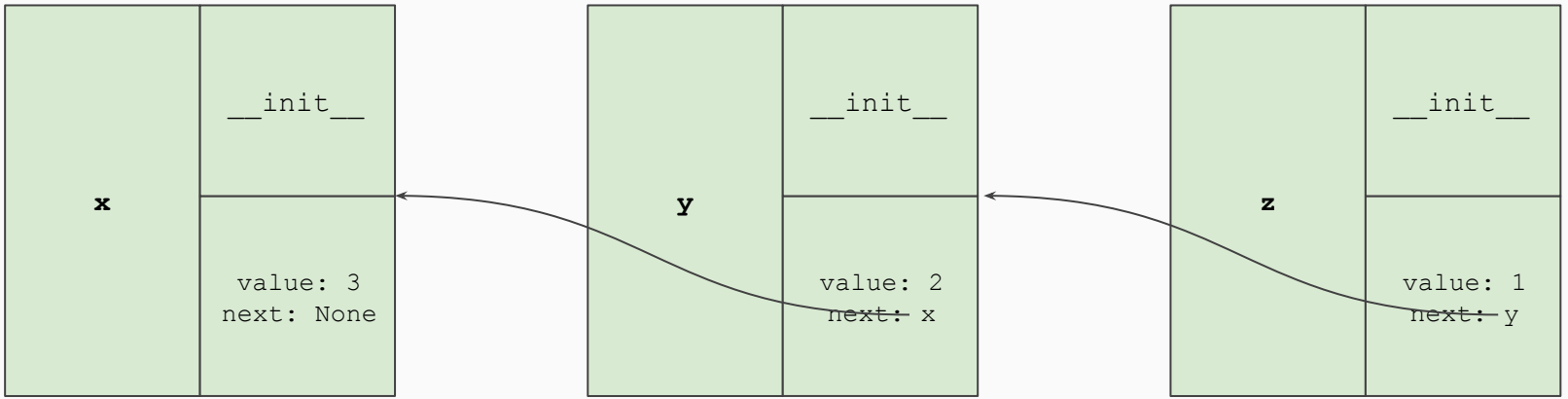
`class`: Start an object recipe definition
`(object)`: **Inherit** from a parent
`__init__`: Define a **constructor** function
`self`: This is a class function, so it takes the object instance as first parameter.
`self.value`: The instance now has an **attribute** called value.



Example: Lists from scratch

```
x = ListItem(3)
y = ListItem(2, x)
z = ListItem(1, y)
```

calls __init__ function



Example: Lists from scratch

```
class List(object):
    def __init__(self, head):
        self.head = head

    def __str__(self):
        temp = self.head
        while temp is not None:
            print(temp.value)
            temp = temp.next
```

List object:

- Assign a reference to the list head item
- Print the value attribute of the head item
- Move temp reference to the next attribute

```
my_list = List(z)
my_list.print_list()
```

Example: Lists from scratch (modified)

```
class List(object):
    def __init__(self, value):
        self.head = ListItem(value)
        self.tail = self.head

    def __str__(self):
        temp = self.head
        while temp is not None:
            print(temp.value)
            temp = temp.next
```

keep a reference to the end

Example: Lists from scratch (special methods)

```
class List(object):
    def __init__(self, value):
        self.head = ListItem(value)
        self.tail = self.head

    def __str__(self):
        temp = self.head
        while temp is not None:
            print(temp.value)
            temp = temp.next
```

rename print list to call special methods

Example: Lists from scratch

```
class List(object):
    ...

    def append(self, value):
        new_item = ListItem(value)
        self.tail.next = new_item
        self.tail = new_item
```

```
my_list = List(1)
my_list.append(2)
my_list.print_list()
```

Example: Lists from scratch

```
class List(object):
    ...

    def len(self):
        len_list = 0
        temp = self.head
        while temp is not None:
            len_list += 1
            temp = temp.next
        return len_list
```

```
print(my_list.len())
```

Example: Lists from scratch

```
class List(object):
    ...
    def getitem(self, index):
        if index >= self.len() or index <
0:
            print("ERROR")
        else:
            temp = self.head
            i = 0
            while i < index:
                temp = temp.next
                i += 1
            return temp.value
```

Example: Lists from scratch - special methods

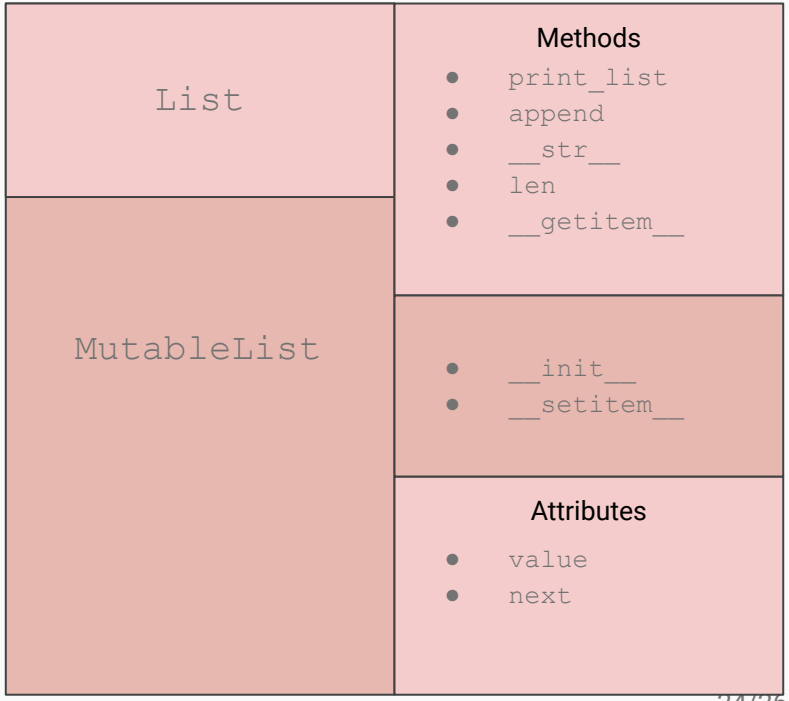
```
class List(object):
    ...
    def __getitem__(self, index):
        if index >= self.len() or index <
0:
            print("ERROR")
        else:
            temp = self.head
            i = 0
            while i < index:
                temp = temp.next
                i += 1
            return temp.value
```

Example: Mutable list

Child of regular List class

```
class MutableList(List):
    def __init__(self, value):
        super().__init__(value)
```

Call parent constructor



Example: Mutable list

```
class MutableList(List):
    ...

    def __setitem__(self, index, value):
        if index >= self.len() or index < 0:
            print("ERROR")
        else:
            temp = self.head
            i = 0
            while i < index:
                temp = temp.next
                i += 1
            temp.value = value
```

Child of regular List class

Call parent constructor

Example: Mutable list

```
class MutableList(List):
    ...

    def __add__(self, other):
        TODO

    def __mul__(self, times):
        TODO using __add__
```

Remember that:

- __add__ stands for (+) and it concatenates two lists:
  [1, 2] + [3, 4] = [1, 2, 3, 4]
- __mul__ stands for (*) and it concatenates a list with itself N times:
  [1, 2] * 3 = [1, 2, 1, 2, 1, 2]