What is Functional Programming?

What is Functional Programming?

Functional programming gives us a way to organize our code, while making sure the code remains easy to test and modify.

What is Functional Programming?

OOP:

“Humans think in terms of objects, therefore programs should be organized in terms of objects”

What is Functional Programming?

FP:

“Computer programs should be as reliable as mathematical functions”

$$f(x) = x + 1$$

What is Functional Programming?

Functional Programming is "declarative":

We focus on *what* things are, instead of *how* to get them

Declarative vs. Imperative

Declarative: “What is a house?”

Imperative: “How do you build a house?”

Declarative vs. Imperative

```
numbers_list = [5, 12, 4, 9, 120]
```

Finding the average of `numbers_list` "imperatively":

1. *Set x equal to zero*
2. *Add the first number in the list to x*
3. *Repeat step 2 for the rest of the numbers in the list*
4. *Divide x by the length of the list*

Declarative vs. Imperative

```
numbers_list = [5, 12, 4, 9, 120]
```

Finding the average of `numbers_list` "declaratively":

"The average is the sum of the numbers in the list, divided by the length of the list"

Declarative vs. Imperative

$$f(x) = x^2 + 5$$

$$f(x) = 3x - 10$$

$$f(x, y) = x + 2y$$

The 3 “Core Concepts” of FP

1. Immutability
2. Separation of Data and Functions
3. First-Class Functions

Core Concept 1: Immutability

Core Concept 1: Immutability

```
x = 5
...
x = 100
...
x = -1
```

Core Concept 1: Immutability

```
x = 5
...
x = 100
...
x = -1
```

Core Concept 1: Immutability

Procedural/OOP treats variables as "buckets" that can hold different values over time

Core Concept 1: Immutability

```
x = 3 -> “x is 3”

pi = 3.14159… -> “pi is 3.14159…”

~~x = 98~~
```

Core Concept 1: Immutability

```
employee_1 =
   new Employee('John', 60000);

employee_1.change_salary(10000);
```

Core Concept 1: Immutability

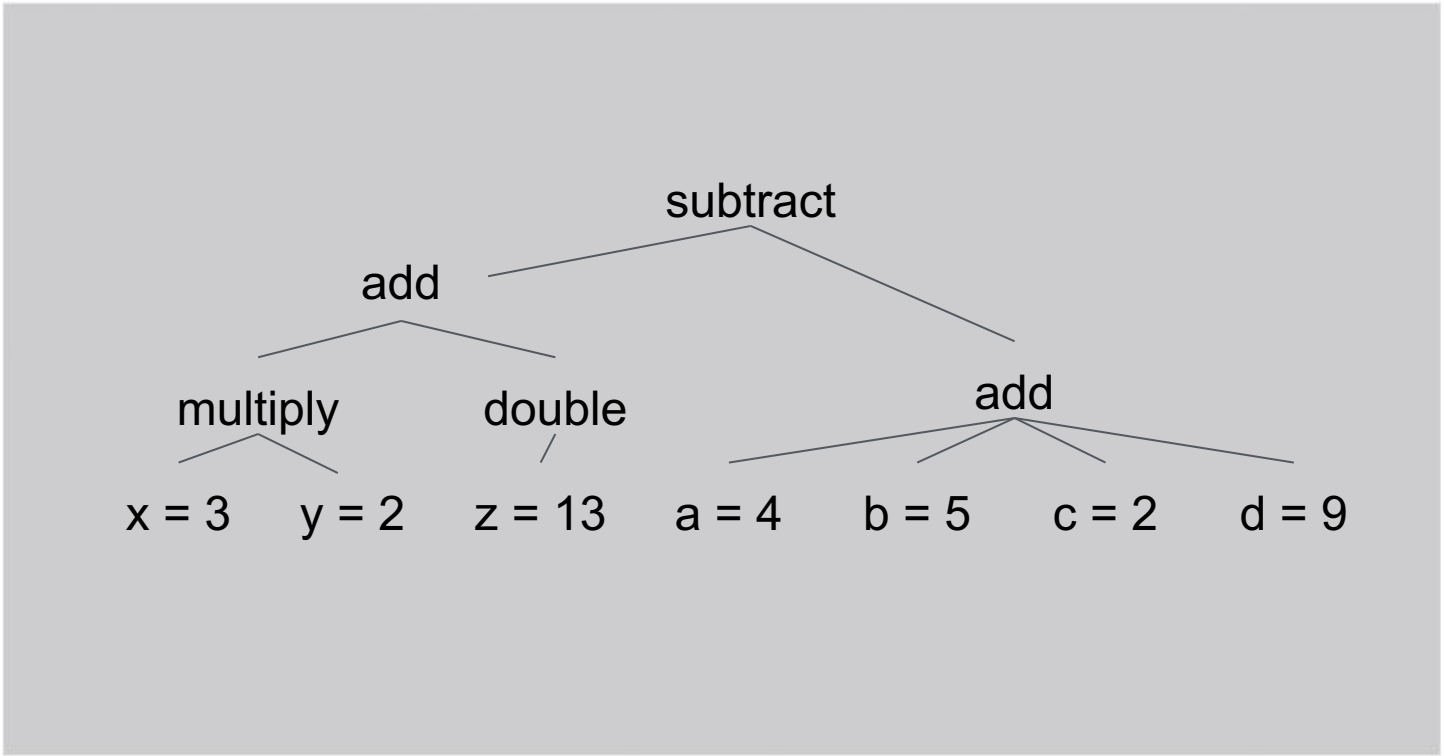
```
employee_1 = {
   'name': 'John',
   'salary': 60000,
}

updated_employee_1 = {
   'name': employee_1.name,
   'salary': 80000,
}
```

Core Concept 1: Immutability

Immutability frees us from dealing with “state change”

Core Concept 1: Immutability



Core Concept 2: Separation of Data and Functions

Core Concept 2: Data/Function Separation

"Data" - any values a program contains

Core Concept 2: Data/Function Separation

"Data" - any values a program contains

"Functions" - operations that we can apply to that data

Core Concept 2: Data/Function Separation

"Data" - any values a program contains

"Functions" - operations that we can apply to that data

Core Concept 2: Data/Function Separation

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

  def increase_age():
    self.age += 1

  def change_name(new_name):
    self.name = new_name
```

Core Concept 2: Data/Function Separation

```
person = {
   'name': 'Sharon',
   'age': 34
}

def change_name(person, new_name):
   ...
```

Core Concept 2: Data/Function Separation

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

  def increase_age():
    self.age += 1

  def change_name(new_name):
    self.name = new_name
```

Core Concept 2: Data/Function Separation

```
class Person:
  def __init__(self, first, last):
    self.first_name = first
    self.last_name = last
    self.initials = f'{first[0]}{last[0]}'
```

Core Concept 2: Data/Function Separation

```
person = Person('William', 'Blake', 41)

person.first_name = 'Bill'
person.initials = 'BB'

person.first_name = 'Will'
# forgot to set initials!
```

Core Concept 2: Data/Function Separation

```
class Person:
  def __init__(self, first, last):
    ...

  def set_first_name(new_name):
    self.__first_name = new_name
    self.__initials =
        f'{new_name[0]}{self.__last_name[0]}'
  ...
```

Core Concept 2: Data/Function Separation

```
def create_person(first, last):
  return {
    'first_name': first,
    'last_name': last,
    'initials': f'{first[0]}{last[0]}'
  }
```

Core Concept 2: Data/Function Separation

```
person = create_person('John', 'Doe')

updated_person =
  create_person('Dwayne', person['last_name'])
```

Core Concept 3: First-Class Functions

Core Concept 3: First-Class Functions

```
list_of_functions = [
  my_function_1,
  my_function_2,
  ...
]

do_something(my_function)

...
return some_function
```

Core Concept 3: First-Class Functions

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


def set_name(new_name):
  self.name = new_name
```

Core Concept 3: First-Class Functions

"Pure" functions always produce the same output given the same input

Core Concept 3: First-Class Functions

```
list_of_functions = [
  my_function_1,
  my_function_2,
  ...
]

do_something(my_function)

...
return some_function
```