Introduction to Functional Programming

Imperative Programming

- Code is written/run sequentially
- Focus on the procedure we take to reach what we want

Imperative Programming

```
def descriptives(iterable):
    """Calcualates and returns descriptive of the iterable"""
    SIZE = len(iterable)
    # Calculating Mean
    total = 0
    for value in iterable:
        total = total + value
    mean = total / SIZE
    # Calculating Variance
    squared_difference_sum = 0
    for value in iterable:
        difference = value - mean
        squared_difference = difference ** 2
        squared_difference_sum = squared_difference_sum + squared_difference
    variance = squared_difference_sum / SIZE
    # Calculating Standard Deviation
    standard_deviation = variance ** (0.5)
    return mean, variance, standard_deviation


descriptives([1, 2, 3, 4, 5, 6])
```

```
(3.5, 2.9166666666666665, 1.707825127659933)
```

Imperative Programming

```
def descriptives(iterable):
    """Calcualates and returns descriptive of the iterable"""
    SIZE = len(iterable)
    # Calculating Mean
    total = 0
    for value in iterable:
        total = total + value
    mean = total / SIZE
    # Calculating Variance
    squared_difference_sum = 0
    for value in iterable:
        difference = value - mean
        squared_difference = difference ** 2
        squared_difference_sum = squared_difference_sum + squared_difference
    variance = squared_difference_sum / SIZE
    # Calculating Standard Deviation
    standard_deviation = variance ** (0.5)
    return mean, variance, standard_deviation


descriptives([1, 2, 3, 4, 5, 6])
```

```
(3.5, 2.9166666666666665, 1.707825127659933)
```

Imperative Programming

```
def descriptives(iterable):
    """Calcualates and returns descriptive of the iterable"""
    SIZE = len(iterable)
    # Calculating Mean
    total = 0
    for value in iterable:
        total = total + value
    mean = total / SIZE
    # Calculating Variance
    squared_difference_sum = 0
    for value in iterable:
        difference = value - mean
        squared_difference = difference ** 2
        squared_difference_sum = squared_difference_sum + squared_difference
    variance = squared_difference_sum / SIZE
    # Calculating Standard Deviation
    standard_deviation = variance ** (0.5)
    return mean, variance, standard_deviation


descriptives([1, 2, 3, 4, 5, 6])
```

```
(3.5, 2.9166666666666665, 1.707825127659933)
```

Imperative Programming

```
def descriptives(iterable):
    """Calcualates and returns descriptive of the iterable"""
    SIZE = len(iterable)
    # Calculating Mean
    total = 0
    for value in iterable:
        total = total + value
    mean = total / SIZE
    # Calculating Variance
    squared_difference_sum = 0
    for value in iterable:
        difference = value - mean
        squared_difference = difference ** 2
        squared_difference_sum = squared_difference_sum + squared_difference
    variance = squared_difference_sum / SIZE
    # Calculating Standard Deviation
    standard_deviation = variance ** (0.5)
    return mean, variance, standard_deviation


descriptives([1, 2, 3, 4, 5, 6])
```

```
(3.5, 2.9166666666666665, 1.707825127659933)
```

Analytics Vidhya

Imperative Programming

- Code is written/run sequentially

Imperative Square

- Code is written/run sequentially

```
def imperative_square():
    for i in range(1, 100000):
        i ** 2


%timeit imperative_square()
```

Analytics Vidhya

Imperative Square

- Code is written/run sequentially

```
def imperative_square():
    for i in range(1, 100000):
        i ** 2


%timeit imperative_square()
```

```
21.3 ms ± 910 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Analytics Vidhya

Imperative Square

- Code is written/run sequentially

```
def imperative_square():
    for i in range(1, 100000):
        i ** 2


%timeit imperative_square()
```

```
21.3 ms ± 910 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

What if we can run iterations in parallel?

Functional Square

```
def imperative_square():
    for i in range(1, 100000):
        i ** 2


%timeit imperative_square()
```

```
21.3 ms ± 910 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
def functional_square():
    map(lambda x: x ** 2, range(1, 100000))


%timeit functional_square()
```

```
388 ns ± 5.22 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Functional Programming

```
def mean(iterable):
    return sum(iterable) / len(iterable)


def variance(iterable):
    MEAN = mean(iterable)
    SIZE = len(iterable)
    squared_deviation = map(lambda x: (x - MEAN) ** 2, iterable)
    squared_deviation_sum = sum(squared_deviation)
    result = squared_deviation_sum / SIZE
    return result


def standard_deviation(iterable):
    return variance(iterable) ** (1 / 2)


def descriptives(function_list, iterable):
    """Runs all the functions in function list over the iterable and returns result list in same order"""
    return list(map(lambda f: f(iterable), function_list))


descriptives([mean, variance, standard_deviation], [1, 2, 3, 4, 5, 6])
```

```
[3.5, 2.9166666666666665, 1.707825127659933]
```

Functional Programming

```
def mean(iterable):
    return sum(iterable) / len(iterable)
```

```
def variance(iterable):
    MEAN = mean(iterable)
    SIZE = len(iterable)
    squared_deviation = map(lambda x: (x - MEAN) ** 2, iterable)
    squared_deviation_sum = sum(squared_deviation)
    result = squared_deviation_sum / SIZE
    return result
```

```
def standard_deviation(iterable):
    return variance(iterable) ** (1 / 2)
```

```
def descriptives(function_list, iterable):
    """Runs all the functions in function list over the iterable and returns result list in same order"""
    return list(map(lambda f: f(iterable), function_list))


descriptives([mean, variance, standard_deviation], [1, 2, 3, 4, 5, 6])
```

```
[3.5, 2.9166666666666665, 1.707825127659933]
```

Functional Programming

```
def mean(iterable):
    return sum(iterable) / len(iterable)


def variance(iterable):
    MEAN = mean(iterable)
    SIZE = len(iterable)
    squared_deviation = map(lambda x: (x - MEAN) ** 2, iterable)
    squared_deviation_sum = sum(squared_deviation)
    result = squared_deviation_sum / SIZE
    return result


def standard_deviation(iterable):
    return variance(iterable) ** (1 / 2)


def descriptives(function_list, iterable):
    """Runs all the functions in function list over the iterable and returns result list in same order"""
    return list(map(lambda f: f(iterable), function_list))
```

```
descriptives([mean, variance, standard_deviation], [1, 2, 3, 4, 5, 6])
```

```
[3.5, 2.9166666666666665, 1.707825127659933]
```

Functional Programming

```
def mean(iterable):
    return sum(iterable) / len(iterable)


def variance(iterable):
    MEAN = mean(iterable)
    SIZE = len(iterable)
    squared_deviation = map(lambda x: (x - MEAN) ** 2, iterable)
    squared_deviation_sum = sum(squared_deviation)
    result = squared_deviation_sum / SIZE
    return result


def standard_deviation(iterable):
    return variance(iterable) ** (1 / 2)


def descriptives(function_list, iterable):
    """Runs all the functions in function list over the iterable and returns result list in same order"""
    return list(map(lambda f: f(iterable), function_list))


descriptives([mean, variance, standard_deviation], [1, 2, 3, 4, 5, 6])
```

```
[3.5, 2.9166666666666665, 1.707825127659933]
```

Functional Programming : Overview

Functional Programming : Overview

- A new Programming Paradigm

Functional Programming : Overview

- A new Programming Paradigm
- Focus : **what to solve**

Functional Programming : Overview

- A new Programming Paradigm
- Focus : **what to solve**
- Uses pure functions

Functional Programming : Overview

- A new Programming Paradigm
- Focus : **what to solve**
- Uses pure functions
- Immutable variables

Functional Programming : Overview

- A new Programming Paradigm
- Focus : **what to solve**
- Uses pure functions
- Immutable variables
- Recursion over Loops

Functional Programming : Overview

- A new Programming Paradigm
- Focus : **what to solve**
- Uses pure functions
- Immutable variables
- Recursion over Loops
- Higher order functions

Functional Programming : Overview

- A new Programming Paradigm
- Focus : **what to solve**
- Uses pure functions
- Immutable variables
- Recursion over Loops
- Higher order functions
- Everything is a mathematical expression

Functional Programming : Overview

- A new Programming Paradigm
- Focus : **what to solve**
- Uses pure functions
- Immutable variables
- Recursion over Loops
- Higher order functions
- Everything is a mathematical expression
- Lazy Evaluation

Thank You