Introduction of Functional Programming for Scientists

Jongsu Kim

Yonsei University

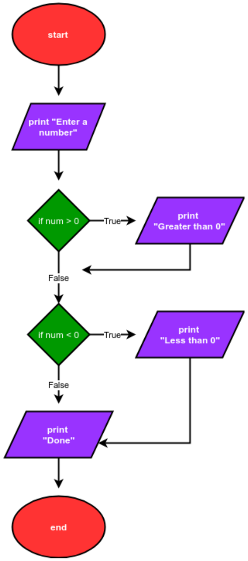
Programming Language Paradigms

- Imperative
  - Procedure Programming
  - Object Oriented Programming
- Declarative
  - Functional Programming
  - ...
- ...

Imperative Programming

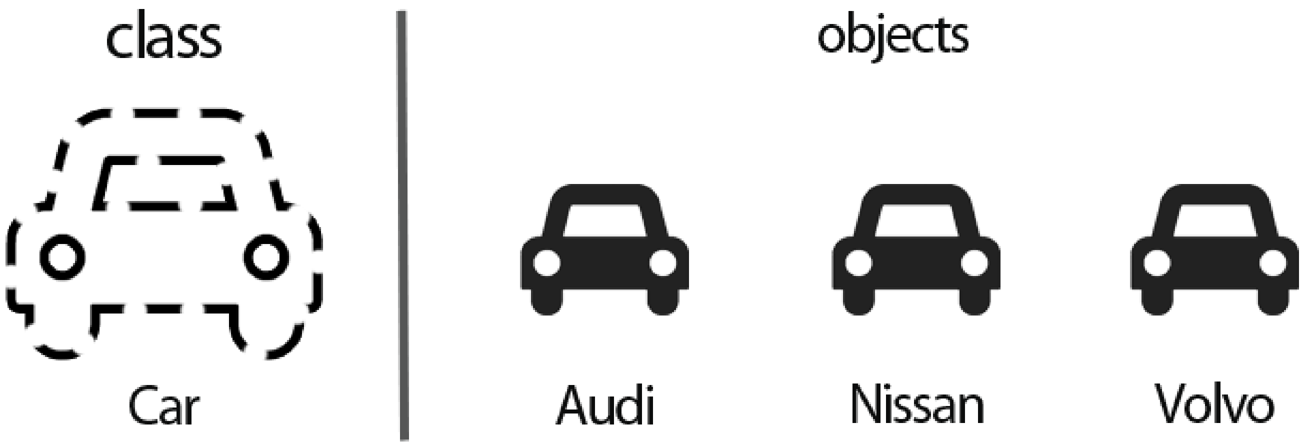
Procedual Programming

- Fortran
- C
- ...

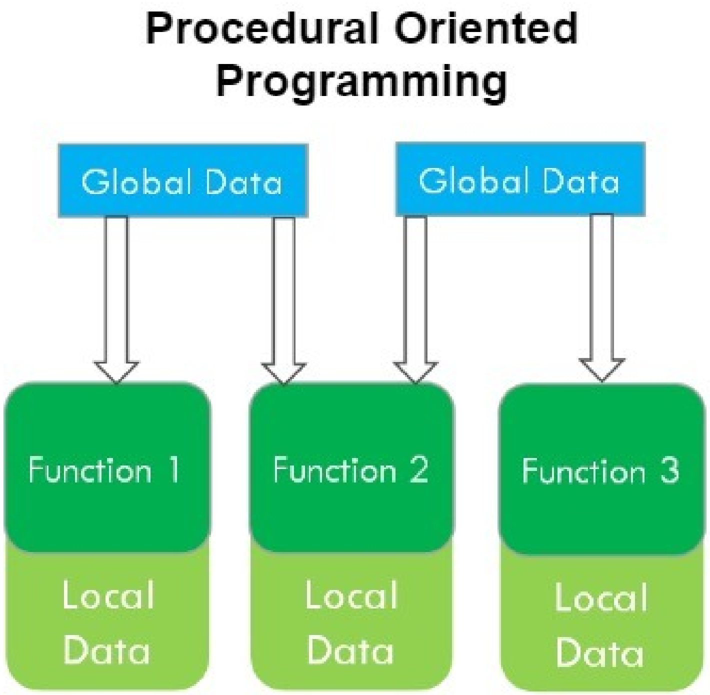


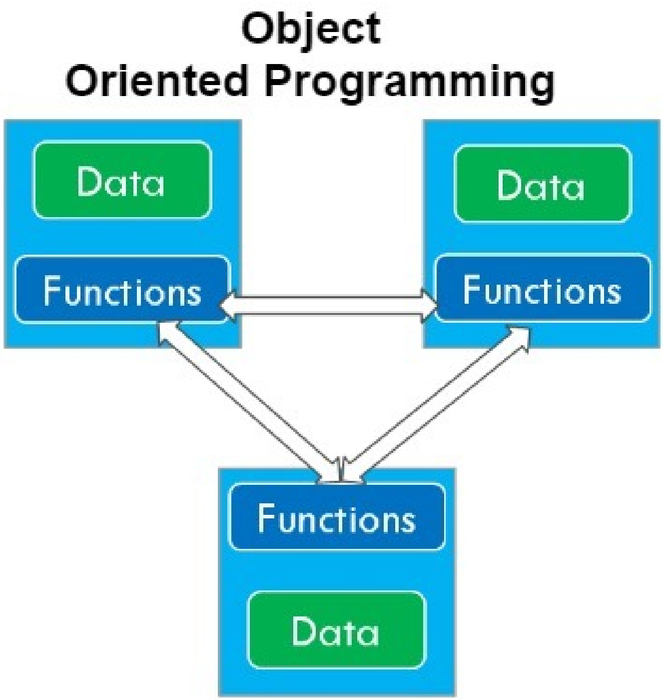
Object Oriented Programming

- C++
- Java
- Python (?)
- ...



Procedure vs OOP





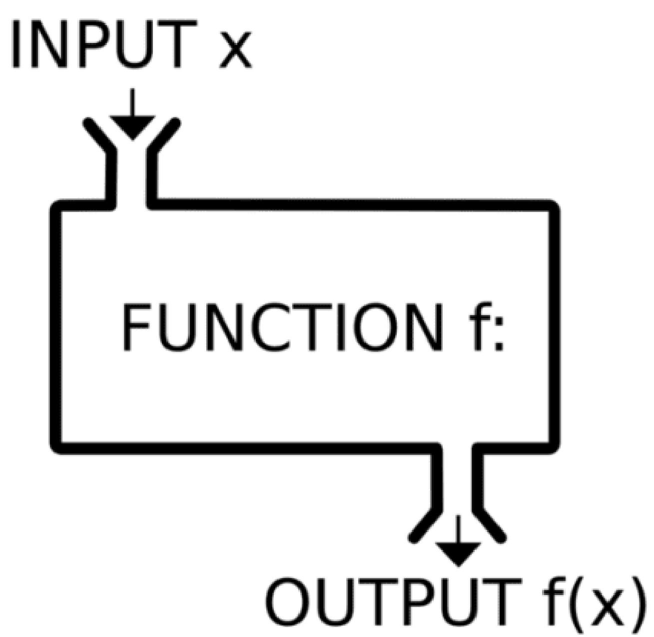
OOP in Tensorflow

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test, verbose=2)
```

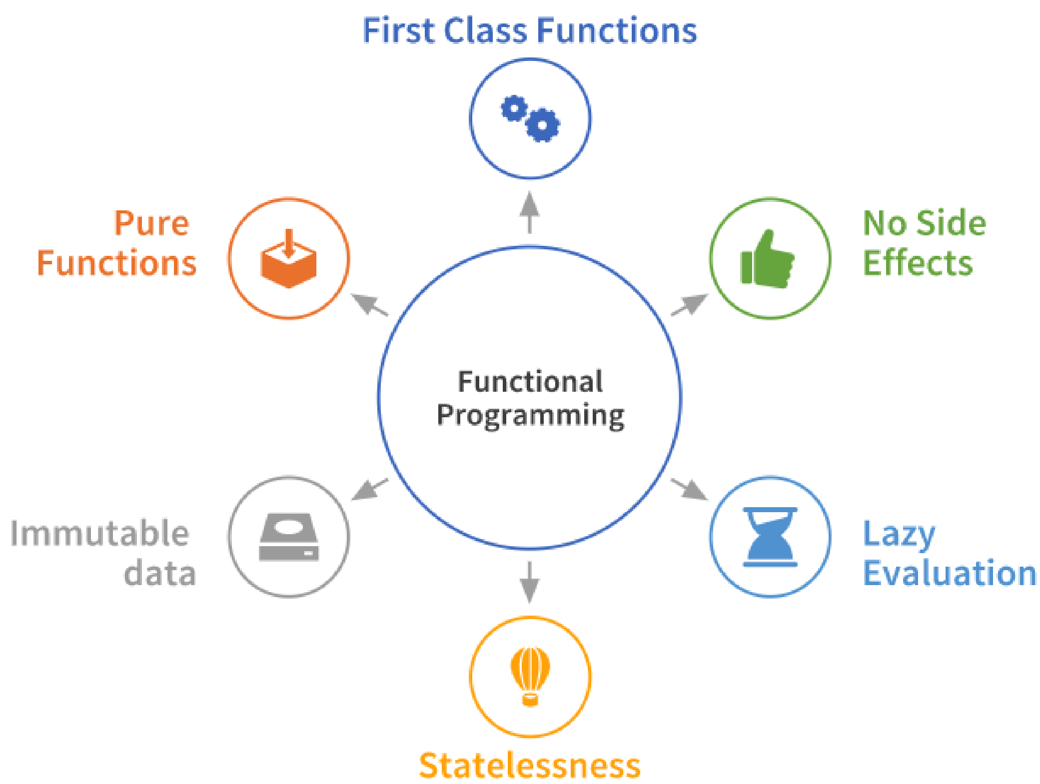
Declarative Programming

Functional Programming

- Lisp
- Haskell
- Clojure
- ...



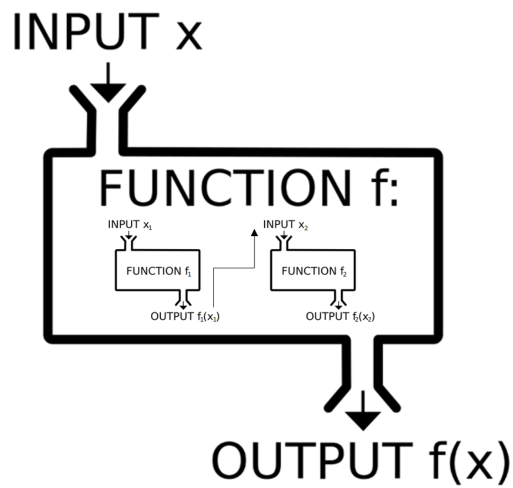
What is Functional Programming?



Pure Function

First Class Function

- Higher order function: Input/Output is a function
- Anonymous and nested function
- Non-local variables and closures



Closure - 1

```
def outer_func():
  x = 5
  def inner_func(y = 3):
    return (x + y)
  return inner_func

a = outer_func()
print(a())
```

Results?

Closure - 1

```
def outer_func():
  x = 5
  def inner_func(y = 3):
    return (x + y)
  return inner_func

a = outer_func()
print(a())
```

Results?

```
8
```

Closure - 2

```
text = "global text"
def outer_func():
  text = "enclosing text"
  def inner_func():
    text = "inner text"
    print('inner_func:', text) # inner_func: global text
  print('outer_func:', text) # outer_func: enclosing text
  inner_func()
  print('outer_func:', text) # outer_func: enclosing text

print('global:', text) # global: global text
outer_func()
print('global:', text) # global: global text
```

Results?

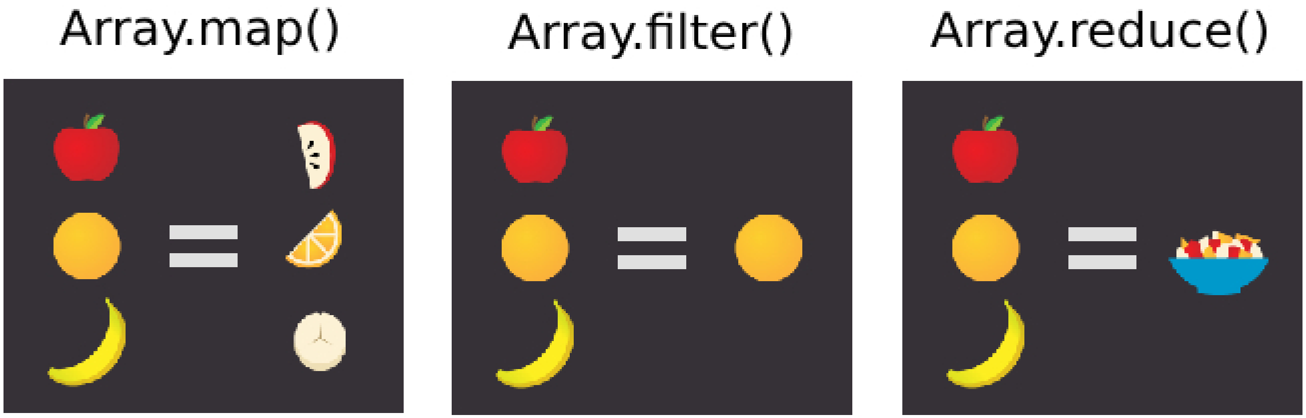
Closure - 2

```
text = "global text"
def outer_func():
  text = "enclosing text"
  def inner_func():
    text = "inner text"
    print('inner_func:', text) # inner_func: global text
  print('outer_func:', text) # outer_func: enclosing text
  inner_func()
  print('outer_func:', text) # outer_func: enclosing text

print('global:', text) # global: global text
outer_func()
print('global:', text) # global: global text
```

Results?

```
global: global text
outer_func: enclosing text
inner_func: inner text
outer_func: enclosing text
global: global text
```

Loops => map/reduce/filter



Why FP? Why now?

Concurrency & Parallelism

Distributed Systems

Optimization

Simplicity

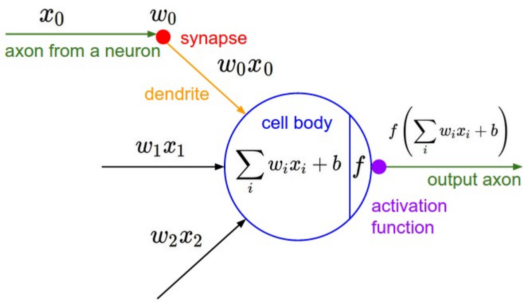
Ceding Control to the Language/Runtime

Focusing on Abstraction

Model becomes Functional

Hide model details in function

```
function compile_model(input_size, output_size)
  unit_size = 16
  model = Flux.Chain(
    Flux.Dense(input_size, unit_size),
    Flux.Dense(unit_size, unit_size),
    Flux.Dense(unit_size, output_size))

  model
end

model = compile_model(input_size, output_size)
y = model(x)
```



More details

```
function compile_model(input_size, output_size, statval)
  unit_size = 16
  model = Flux.Chain(
    Flux.Dense(input_size, unit_size),
    Flux.Dense(unit_size, unit_size),
    Flux.Dense(unit_size, output_size))

  loss(x, y) = Flux.mse(model(x), y)
  # statval : collection of stats such as μ and σ
  # :RMSE : symbol for evaluation metric indicator
  accuracy(data) = evaluation(data, model, statval, :RMSE)
  opt = Flux.ADAM()

  model, loss, accuracy, opt
end

model, loss, accuracy, opt = compile_model(input_size, output_size)
train!(loss, Flux.params(model), train_set, opt)
@show loss(x, y)
@show accuracy(test_set)
```

How evaluation works

Elementry Functions

```
# column-wise sum
_RMSE(y::AbstractVector, ŷ::AbstractVector) =
  sqrt.(sum((y .- ŷ).^2, dims=[1]) / length(y))
_RMSE(y::AbstractMatrix, ŷ::AbstractMatrix) =
  sqrt.(sum((y .- ŷ).^2, dims=[1]) / size(y, 1))
```

Abstraction for any dataset type

```
function RMSE(dataset, model, statvals::AbstractNDSparse)
  _μ = statvals["total", "μ"][:value]
  _σ = statvals["total", "σ"][:value]

  # sum(f, itr) + do block
  # no allocation + mapreduce => faster!
  let cnt = 0
    # column sum for batches
    _rmsesum = sum(dataset) do xy
      _rmseval = _RMSE(xy[2], model(xy[1]))

      # replace Inf, NaN to zero, no impact on sum
      replace!(_rmseval, Inf => 0, -Inf => 0, NaN => 0)
      # number of columns which is not Inf and not NaN
      cnt += count(x -> !(isnan(x) || isinf(x)), _rmseval)

      _rmseval
    end

    # rmse mean
    _rmsesum / cnt
  end
end
```

Abstraction for any single metric

```
function evaluation(dataset, model, statvals::AbstractNDSparse, met
  eval(quote
    let _cnt = 0
      # column-wise sum for batches
      _sum = sum($(dataset)) do xy
        # Float or (1 x batch) Array
        _val = $(metric)(xy[2], $(model)(xy[1]))

        # number of columns which is not Inf and not NaN
        # if _val is Float, cnt must be 1
        _cnt += count(x -> !(isnan(x) || isinf(x)), _val)

        # If _val is Array -> use `replace!` to replace Inf, NaN to
        # If _val is Number -> just assign zero
        typeof(_val) <: AbstractArray ? replace!(_val, Inf=> 0, -In
          zero(_val)))

        sum(_val)
      end # do - end

      _cnt = _cnt == 0 && isapprox(_sum, 0.0) ? 1 : _cnt

      # mean
      _sum / _cnt
    end # let - end
```

Evaluate over mulitple metrics

```
function evaluations(dataset, model, statvals::AbstractNDSparse, metr
  metric_vals = map(metrics) do metric
    evaluation(dataset, model, statvals, metric)
  end # do - end

  Tuple(metric_vals)
end
```

Evaluate over mulitple metrics

```
function evaluations(dataset, model, statvals::AbstractNDSparse, metr
  metric_vals = map(metrics) do metric
    evaluation(dataset, model, statvals, metric)
  end # do - end

  Tuple(metric_vals)
end
```

Usage

```
eval_metrics = [:RMSE, :MAE, :MSPE, :MAPE]
# test set
for metric in eval_metrics
  _eval = evaluation(test_set, model, statval, metric)
  @info " $(string(ycol)) $(string(metric)) for test : ", _eval
end
```