A Few Function Programing Principles Every Engineer Should Know

What's Functional Programing?

> In computer science, functional programming is **a programming paradigm** that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. [1]

[1] Wikipedia - Functional Programining

What's Functional Programing?

What's programing paradigm?

A style of building the structure and elements of computer programs.
Common paradiagm includes [1]:

- Imperative
- Declarative
- Object-oriented
- Functional

Note: Paradigm can overlap or can be used w/ each other.

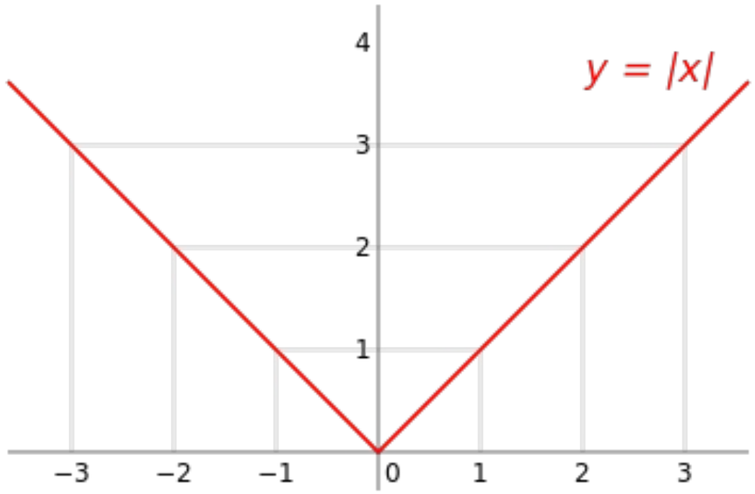
[1] Definition of Programing Paradigms

FP Principles

- Treat computation as the evaluation of math functions
- Pure Function
- High-order Function
- Avoid Mutation
- ...

Treat computation as the evaluation of math functions

A math function maps a set of input to a set of output.



FP Principles

Treat computation as the evaluation of math functions

FP usually prefers expression over statement

```
# Assume val is int

# Statement
def abs1(val):
  if val < 0:
    return -val
  else:
    return val

# Expression
def abs2(val):
  return -val if val < 0 else val
```

Treat computation as the evaluation of math functions

FP usually prefers recursion over iteration

```
# Iteration - Procedural Style
def sum1(nums):
  total = 0
  for num in nums:
    total += num
  return total

# Recursion - Functional Style
def sum2(nums):
   # Not very efficient in Python but you got the idea
   return 0 if len(nums) == 0 else nums[0] + sum2(nums[1:])
```

FP Principles

Pure Function

1. No side effect
2. Same input, same output

Pure Function

#1 No side effect

Example of Function w/ Site Effect

```
def sayhi(name)
  print("Hi there, my name is %s" % name)

def write_to_file(file, text):
  with open(file, 'w+') as f:
    f.write(text)

def sort_nums(nums):
  nums.sort() # inplace sort
  return nums
```

FP Principles

Pure Function

#1 No side effect

But why create an app that can't change the "world"?

> In practice, applications need to have some side effects. Simon Peyton-Jones, a major contributor to the functional programming language Haskell, said the following: "In the end, any program must manipulate state. A program that has no side effects whatsoever is a kind of black box. All you can tell is that the box gets hotter." The key is to limit side effects, clearly identify them, and avoid scattering them throughout the code. [1]

[1] Why are side-effects considered evil in functional programming?

Pure Function

#2 Same input, same output

Example of Function that Violates this

```
gloabl_counter = 0

function foo(x){
  gloabl_counter += 1;
  return gloabl_counter + x;
}

foo(1) // => 2
foo(1) // => 3
```

Other examples?

FP Principles

High-order Function

1. Take at least one function as input or
2. Return a function

FP Principles

High-order Function

Example: A function that takes another function as input

```
nums = [1,2,3]

map(lambda x: x*2, nums)
# => [2,4,6]

filter(lambda x: x % 2 == 0, nums)
# => [2]

reduce(lambda total, num: total * num, nums)
# => 6
```

High-order Function

Example: A function that returns a function as output

```
dirs = ['lib', 'bin', 'log']

# `functools` is a built-in Python module
# `functools.partial` return a partial-applied func
func = functools.partial(os.path.join, '/User/jason')

map(func, dirs)
# => ['/User/jason/lib', '/User/jason/bin', '/User/jason/log']
```

High-order Function

Q: Can you do this w/ just lambda?

```
map(lambda dirname: os.path.join('/User/jason', dirname), dirs)
# => ['/User/jason/lib', '/User/jason/bin', '/User/jason/log']
```

High-order Function

Another Example -- Closure

```
def compose(f, g):
  def composed_func(val):
    return g(f(val))
  return composed_func
```

```
dirname = '~/lib/../log'

os.path.abspath(os.path.expanduser(dirname))
# => /Users/jiamingz/log

expandpath = compose(os.path.expanduser, os.path.abspath)
expandpath(dirname)
expandpath('~/lib/python/..')
```

High-order Function

Closure is a common high-order function in FP lang

Avoid Mutation

Wait ... what about the efficiency?

Persistent data structure

Not gonna cover to this time. If you're interested, check this Ideal Hash Tree