Introduction to Functional Programming in Python

Remark

- Techniques from F#.

About speaker

- Chang HaiBin
- Data Engineer at a Hedge Fund
- M.Sc. Uni. of Michigan (Math)

- Financial Engineer (Numerical Technologies)
- Business Analyst (U.S. Mattress)

3 concepts

3 concepts:

• keep	(filter)
• change	(map)
• then	(pipe-forward)

- Omitted: “reduce”

keep (filter)

keep (filter)

- From 1 to 10, keep even numbers (remove odd numbers)
- What is the result?

keep (filter)

- From 1 to 10, keep even numbers (remove odd numbers)
- `[2,4,6,8,10]`

keep (filter)

- From 1 to 10, keep prime numbers (remove non-prime)
- What is the result?

keep (filter)

- From 1 to 10, keep prime numbers (remove non-prime)
- `[2,3,5,7]`

keep (filter)

- Given a list, and a criteria/condition,
- Create a new, shorter list
- Keep those that are True
- (remove those that are False)

Remark

- You can also define the “remove” function, that does the opposite of “keep” change (map)

change (map)

- From 1 to 10, change each x to (x * x)
- What is the result?

change (map)

- From 1 to 10, change each x to (x * x)

- `[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]`

change (map)

- From 1 to 10, change each x to (1 / x)

- What is the result?

change (map)

- From 1 to 10, change each x to (1 / x)

- `[1/1,    1/2,    1/3,    1/4,    1/5,`
  `1/6,    1/7,    1/8,    1/9,    1/10]`

change (map)

- From 1 to 10, change each x to (1 / x)

-
  ```
  [1,      0.5,     0.3333, 0.25,  0.2,
  0.1666, 0.1428, 0.125, 0.1111, 0.1]
  ```

change (map)

- Given a list, and a formula to change each element,
- Create a new list, where each result depends on the original list and the formula

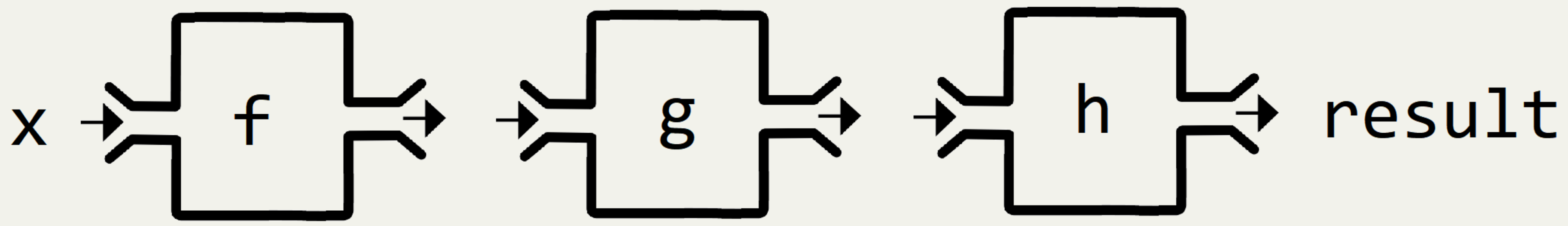
Why?

- “keep” and “change” allows you to avoid for-loop.
  - Or at least, use for-loop implicitly.

- Allows you to work on a higher level (no need to specify details of the loop)

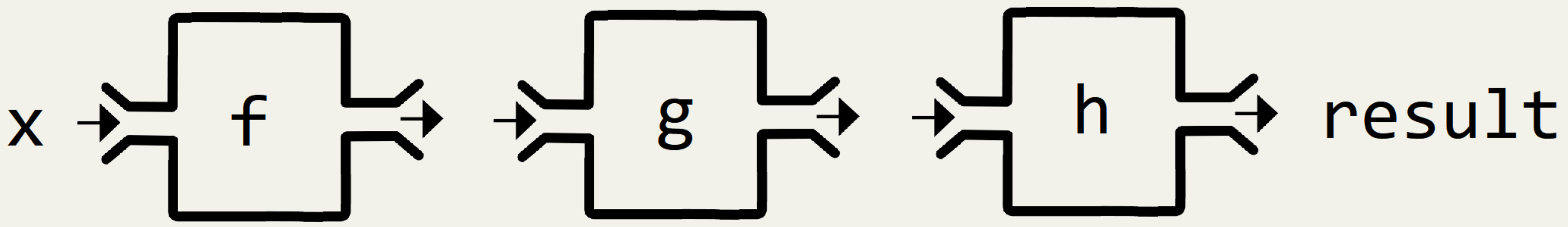
then (pipe-forward)

then



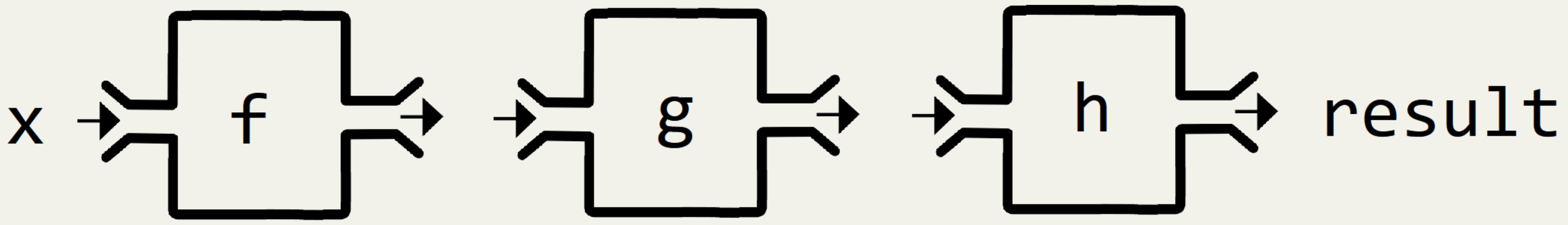
then

- `h(g(f(x)))`



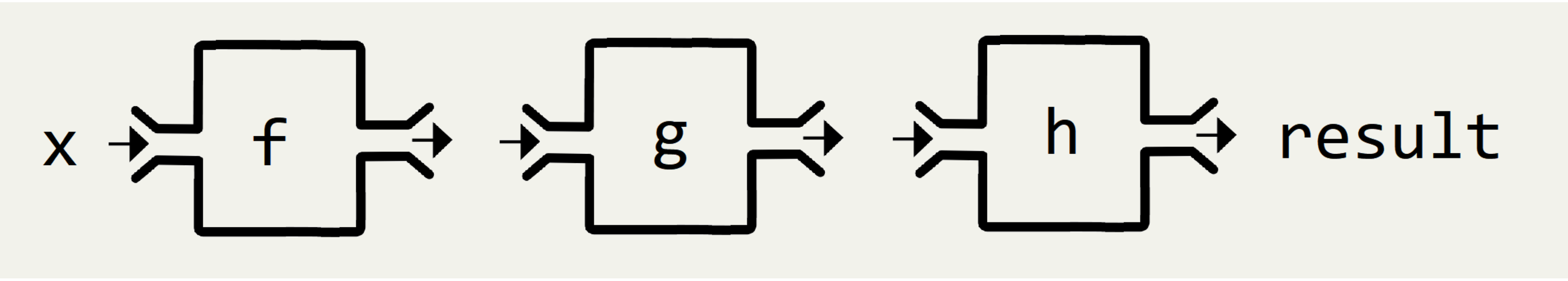
then

- `0. Use x`
- `1. Do f`
- `2. Do g`
- `3. Do h`



then

```
x   \
  | then | f \
  | then | g \
  | then | h
```



How to define “then”

```
from functools import partial
class Infix(object):
    def __init__(self, func):
        self.func = func
    def __or__(self, other):
        return self.func(other)
    def __ror__(self, other):
        return Infix(partial(self.func, other))
    def __call__(self, v1, v2):
        return self.func(v1, v2)

then = Infix(lambda x,f: f(x))
```

(Demo) Project Euler

- Math/Programming Challenge problems.

Question 1

- From 1 to 999, find the sum of all numbers that are either multiples of 3, or multiples of 5.

Solution Q1

```
range(1,1000) \
| then | keep(lambda x : x % 3 == 0 or x % 5 == 0) \
| then | sum \
| then | print
```

Solution Q1

```
range(1,1000) \
| then | keep(lambda x : x % 3 == 0 or x % 5 == 0) \
| then | sum \
| then | print
```

```
Start from 1 to 999
Then keep the numbers you want (multiples of 3 or 5)
Then sum up the remaining numbers
Then print the result
```

```
range(1,1000)
```

```
[1, 2, 3, 4, 5, ......, 999]
```

```
range(1,1000) \
| then | keep(lambda x : x % 3 == 0 or x % 5 == 0)
```

```
[3, 5, 6, 9, 10, 12, 15, 18, 20, ......]
```

```
range(1,1000) \
| then | keep(lambda x : x % 3 == 0 or x % 5 == 0) \
| then | sum
```

```
233168
```

```
range(1,1000) \
| then | keep(lambda x : x % 3 == 0 or x % 5 == 0) \
| then | sum \
| then | print
```

```
Print 233168 to console
```

Question 2

- In the Fibonacci numbers:
- 1,2,3,5,8,13,21,34,55,......

- What is the sum of even numbers less than 4 million (in the Fibonacci numbers)?

Solution Q2

```
baseList \
| then | keep(lambda x : x < 4000000) \
| then | keep(lambda x : x % 2 == 0) \
| then | sum \
| then | print
```

Remark: Extra details in construction of baseList.

Solution Q2

```
baseList
```

```
[1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...................]
```

Solution Q2

```
baseList \
| then | keep(lambda x : x < 4000000)
```

```
[1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ......, 3524578]
```

Solution Q2

```
baseList \
| then | keep(lambda x : x < 4000000) \
| then | keep(lambda x : x % 2 == 0)
```

```
[2, 8, 34, ......, 3524578]
```

Solution Q2

```
baseList \
| then | keep(lambda x : x < 4000000) \
| then | keep(lambda x : x % 2 == 0) \
| then | sum
```

```
4613732
```

Solution Q2

```
baseList \
| then | keep(lambda x : x < 4000000) \
| then | keep(lambda x : x % 2 == 0) \
| then | sum \
| then | print
```

```
Print 4613732 to Console
```

Question 4

- A palindromic number reads the same both ways. e.g. “14241”, “927729”

- Which two 3-digit numbers, “a” and “b”, will create the largest palindrome product:
- c = a * b

Solution Q4

```
number_pairs \
| then | change(lambda a,b: a * b) \
| then | keep(IsPalindrome) \
| then | max \
| then | print
```

```
Remark: Need slight modification for functions accepting
Tuples.
```

Question 6

- Calculate:

  $(1 + 2 + ... + 100)^2 \quad - (1^2 + 2^2 + ... + 100^2)$

$$(1 + 2 + \ldots + 100)^2$$

```
LHS = \
    range(1,101) \
    | then | sum \
    | then | (lambda s: s * s)
```

$$-(1^2 + 2^2 + \ldots\ldots + 100^2)$$

```
RHS = \
    range(1,101) \
    | then | change(lambda x : x * x) \
    | then | sum
```

Solution Q6

```
RHS = \
    range(1,101) \
    | then | change(lambda x : x * x) \
    | then | sum
LHS = \
    range(1,101) \
    | then | sum \
    | then | (lambda s: s * s)

print(LHS - RHS)
```

Question 8

- 821663704844031**9989**0008895243450658541227588666881
- The 4 neighboring digits with the largest product is 9 x 9 x 8 x 9 = 5832

- Given a 1000-digit number*, find the largest product created by 13 adjacent digits.

- * https://projecteuler.net/problem=8

Solution Q8

```
long_string \
| then | change (int) \
| then | windowed(13) \
| then | change(product) \
| then | max \
| then | print
```

Solution Q8

```
long_string \
| then | change (int)
```

```
[8,2,1,6,6,3,7,0,4,8,4,4,0,.............]
```

Solution Q8

```
long_string \
| then | change (int) \
| then | windowed(13)
```

```
[[8,2,1,6,6,....], [2,1,6,6,.......], [1,6,6,..........],
......]
```

Solution Q8

```
long_string \
| then | change (int) \
| then | windowed(13) \
| then | change(product)
```

```
[    product1      ,      product2      ,      product3      ,
......]
```

Solution Q8

```
long_string \
| then | change (int) \
| then | windowed(13) \
| then | change(product)  \
| then | max
```

```
max_product
```

Solution Q8

```
long_string \
| then | change (int) \
| then | windowed(13) \
| then | change(product)  \
| then | max \
| then | print
```

```
print  max_product  to the Console.
```

Question 9

- Find the positive integers, a,b,c, that satisfy:
  - $a < b < c$
  - $a + b + c = 1000$
  - $a^2 + b^2 = c^2$

Solution Q9

```
all_pairs \
| then | change(lambda a,b: (a,b,1000 - a - b)) \
| then | keep(lambda a,b,c: a < b < c) \
| then | keep(lambda a,b,c: a * a + b * b == c * c) \
| then | print
```

Remark: Need modification for functions accepting Tuples.

Non-math example 1

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’” \
  | then | today.strftime \
  | then | sqlDB.fetchall \
  | then | change(lambda sales: sales.country.upper()) \
  | then | keep(lambda sales: sales.country = ‘SG’) \
  | then | sumby(lambda sales: sales.amount) \
  | then | (lambda totalsale: totalsale * 0.2)
```

Remark: Demonstration only. Actual code may differ.
(e.g. when using upper())

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’”
```

```
“select * from salesDB where date = ‘%Y-%m-%d’”
```

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’” \
  | then | today.strftime
```

```
“select * from salesDB where date = ‘2019-07-08’”
```

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’” \
  | then | today.strftime \
  | then | sqlDB.fetchall
```

```
[($200, ‘SG’), ($160, ‘my’), ($300, ‘sg’), ......]
```

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’” \
  | then | today.strftime \
  | then | sqlDB.fetchall \
  | then | change(lambda sales: sales.country.upper())
```

```
[($200, ‘SG’), ($160, ‘MY’), ($300, ‘SG’), ......]
```

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’” \
  | then | today.strftime \
  | then | sqlDB.fetchall \
  | then | change(lambda sales: sales.country.upper()) \
  | then | keep(lambda sales: sales.country = ‘SG’)
```

```
[($200, ‘SG’), ($300, ‘SG’), ($500, ‘SG’) ]
```

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’” \
  | then | today.strftime \
  | then | sqlDB.fetchall \
  | then | change(lambda sales: sales.country.upper()) \
  | then | keep(lambda sales: sales.country = ‘SG’) \
  | then | sumby(lambda sales: sales.amount)
```

```
$1000
```

```
commission = \
  “select * from salesDB where date = ‘%Y-%m-%d’” \
  | then | today.strftime \
  | then | sqlDB.fetchall \
  | then | change(lambda sales: sales.country.upper()) \
  | then | keep(lambda sales: sales.country = ‘SG’) \
  | then | sumby(lambda sales: sales.amount) \
  | then | (lambda totalsale: totalsale * 0.2)
```

```
$200
```

Non-math example 2

```
data_count = \
  local_folder \
  | then | os.listdir \
  | then | remove(lambda name: name in bad_set) \
  | then | change(count_num_lines) \
  | then | sum
```

(Optional Topic) `reduce`

(Optional) reduce

```
start = 0
for x from 1 to 5:
    start = start + x
print(start)

# 15
```

(Optional) reduce

```
start = 1000
for x from 1 to 5:
    start = start + x
print(start)

# 1015
```

Starting value matters!

(Optional) reduce

```
start = 1000
for x from 1 to 15:
    start = start + x
print(start)

# 1120
```

Range of value matters!

(Optional) reduce

```
start = 1000
for x from 1 to 15:
    start = start * x
print(start)

# 1307674368000000
```

Formula matters!

(Optional) reduce

Given:

1. A starting value
2. A list of elements
3. A transition formula that updates the starting value

We can calculate the final value after accumulating over the whole list.

(Optional Topic) Infix Operator

```
3 + 4
```

The + symbol connects the left number and right number

```
3 * 4
```

The * symbol connects the left number and right number

```
exp1   | then |   exp2
```

```
The | then | symbol connects the left
expression and right expression
```

```
 exp1
| then |   exp2
```

The | then | symbol connects the top expression and bottom expression

Summary

• keep	(filter)
• change	(map)
• then	(pipe-forward)

Q&A