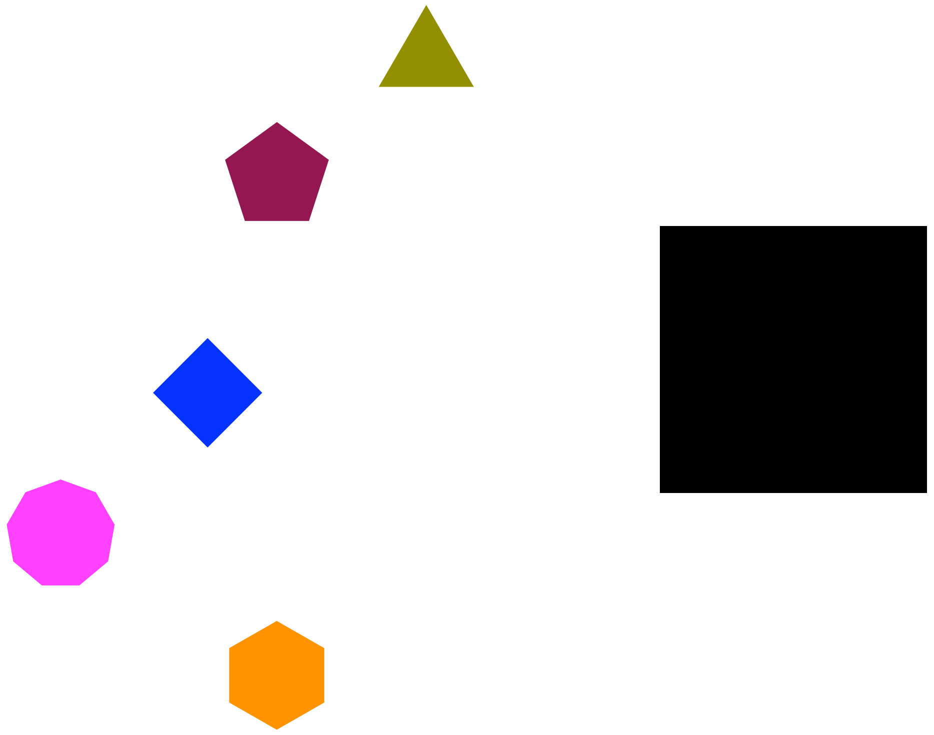
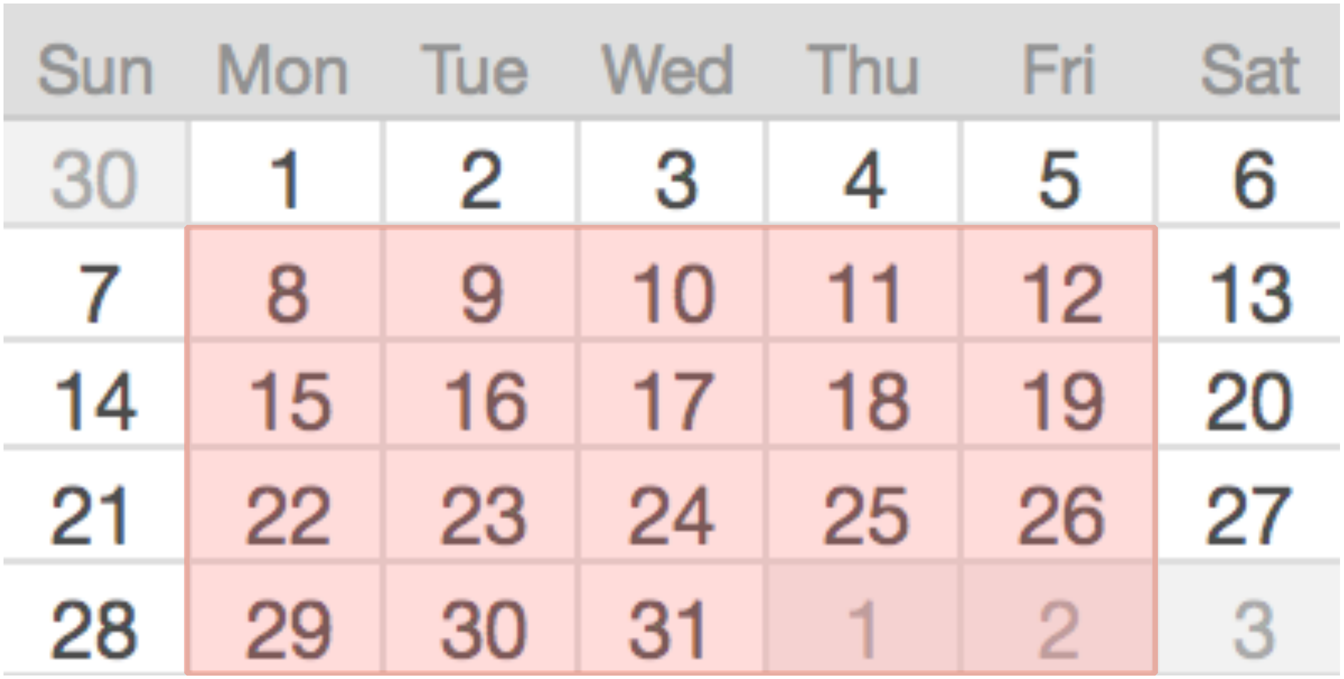
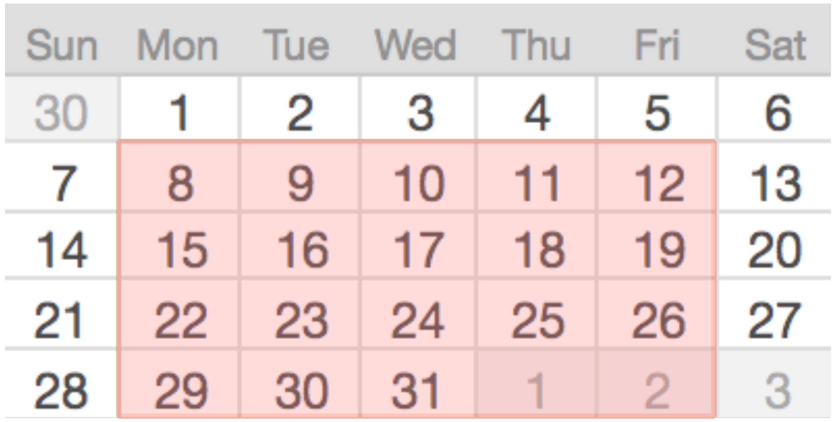
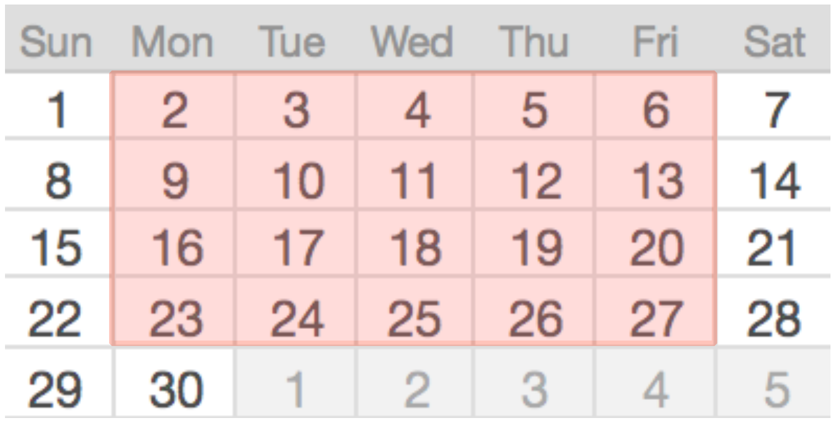




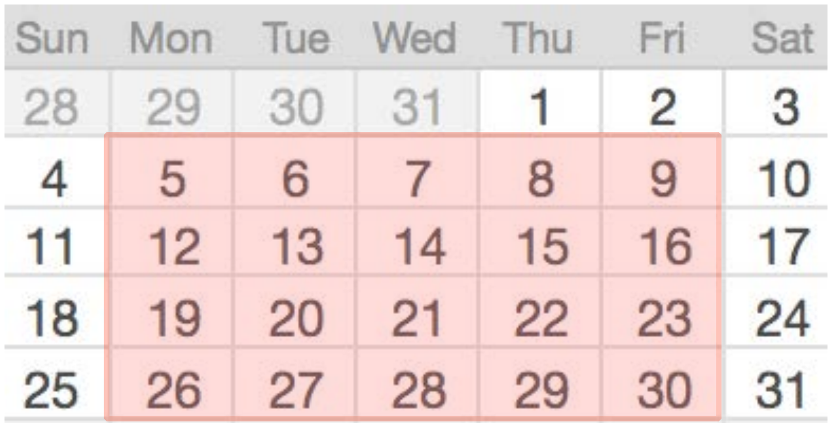
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3



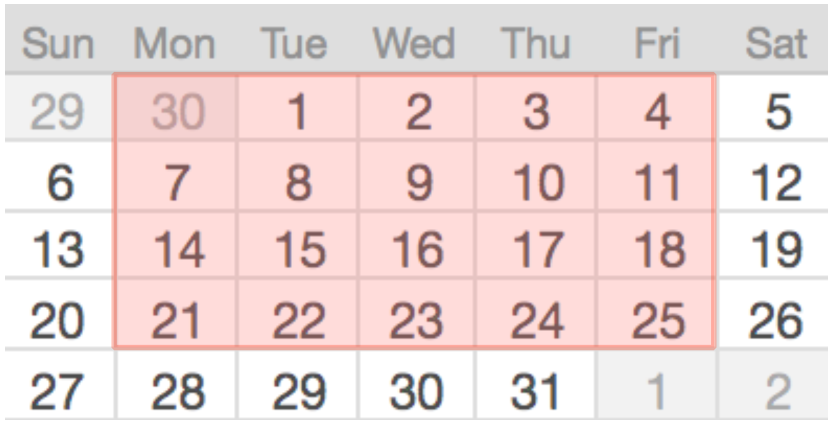
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3



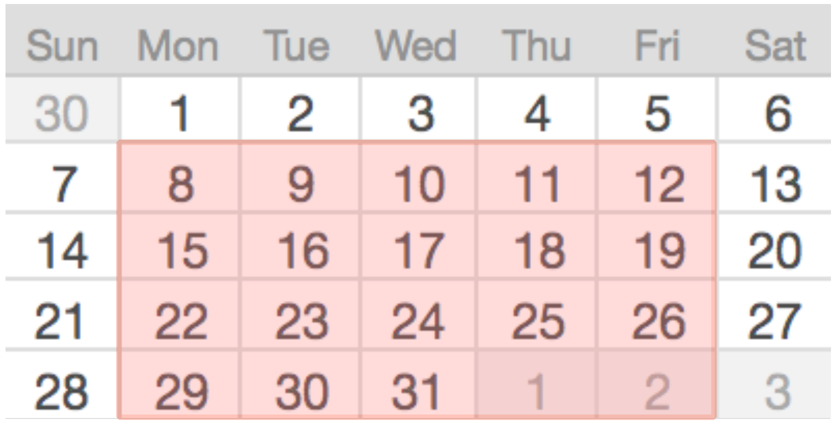
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5



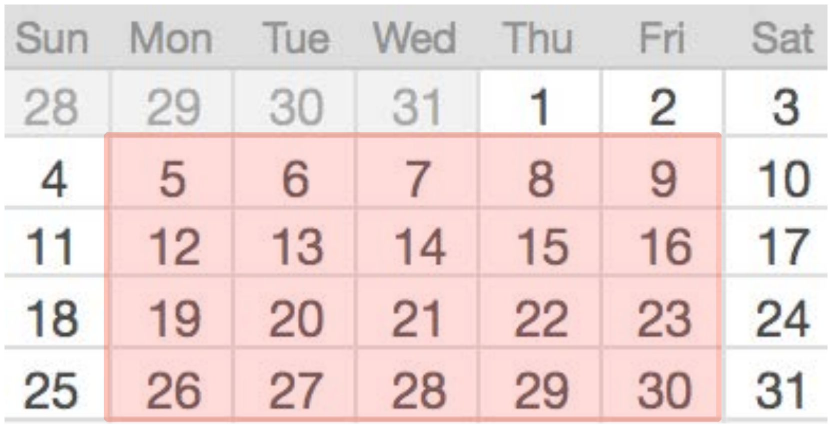
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31



Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2



Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3



Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Welcome to functional programming.

programming =

Thinking

+

Coding

functional programming =

functional Thinking

+

functional Coding

Get Java 8 at
http://jdk8.java.net/download.html

Java 8

This course was created with an
Early Access edition of Java 8.

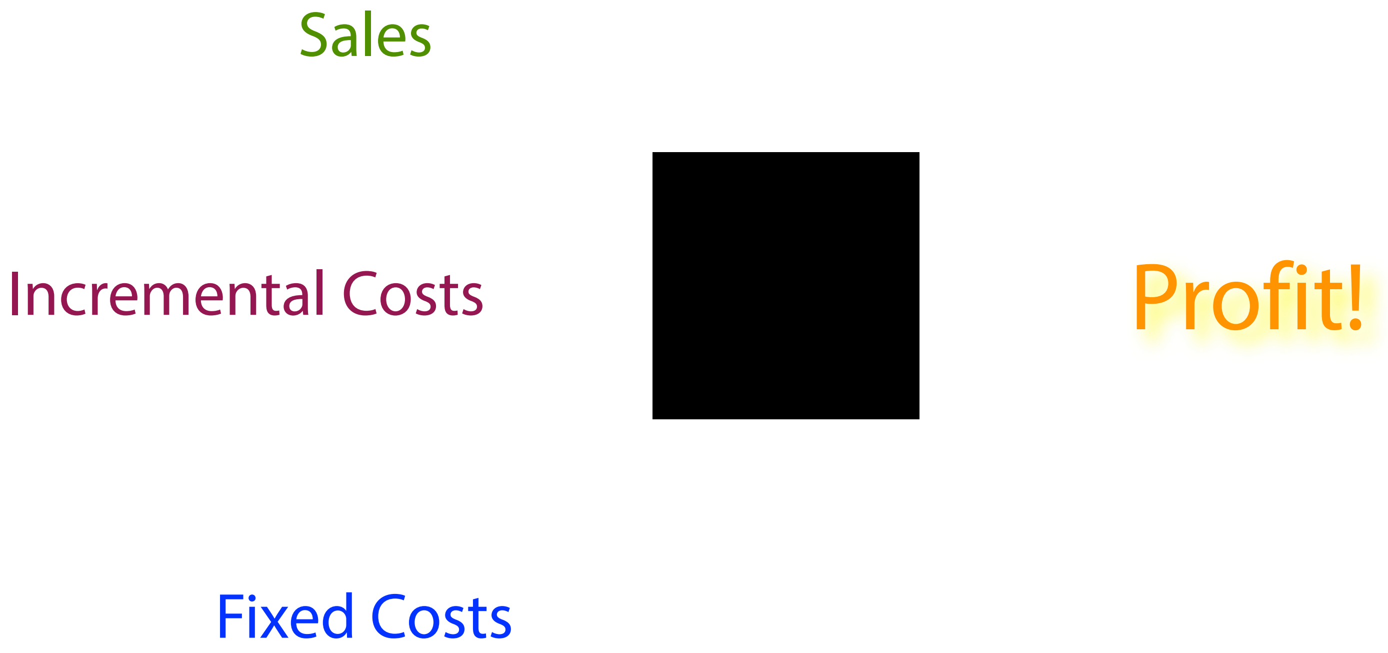
Java 6

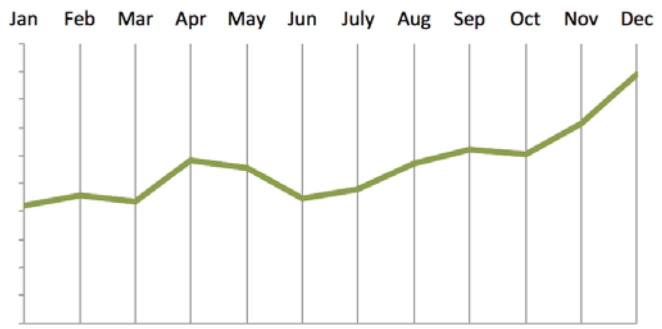
Functions as Values

STORE functions in variables

PASS functions in parameters

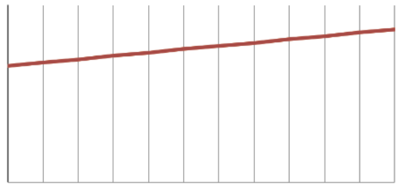
RETURN functions from other functions


Sales
Incremental Costs
Fixed Costs
Profit!

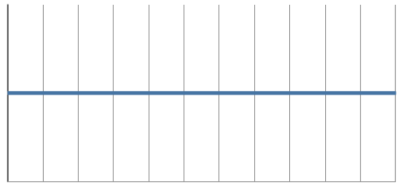


Sales

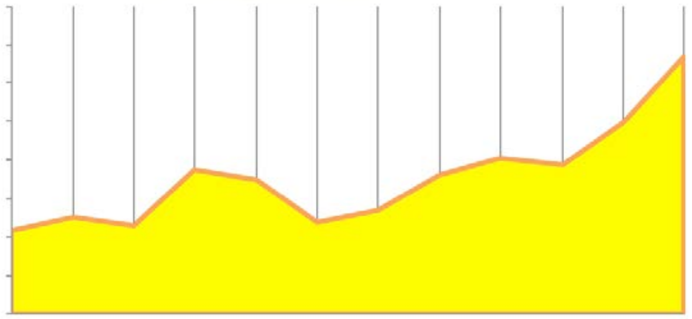
Incremental Costs



Fixed Costs



Profit!



lambda

Functions:

Meaning

evaluation has no outside effect

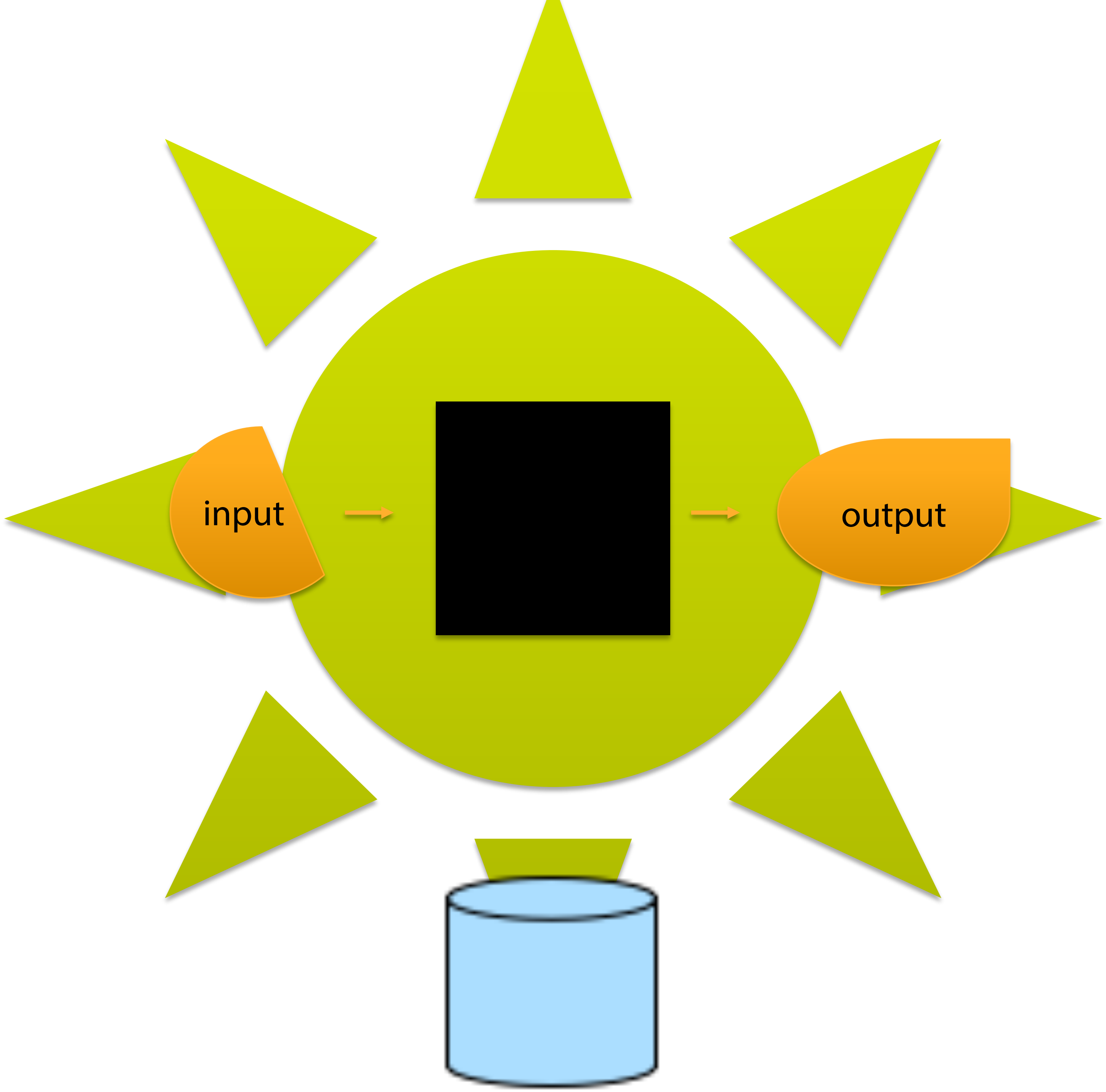
Mechanism

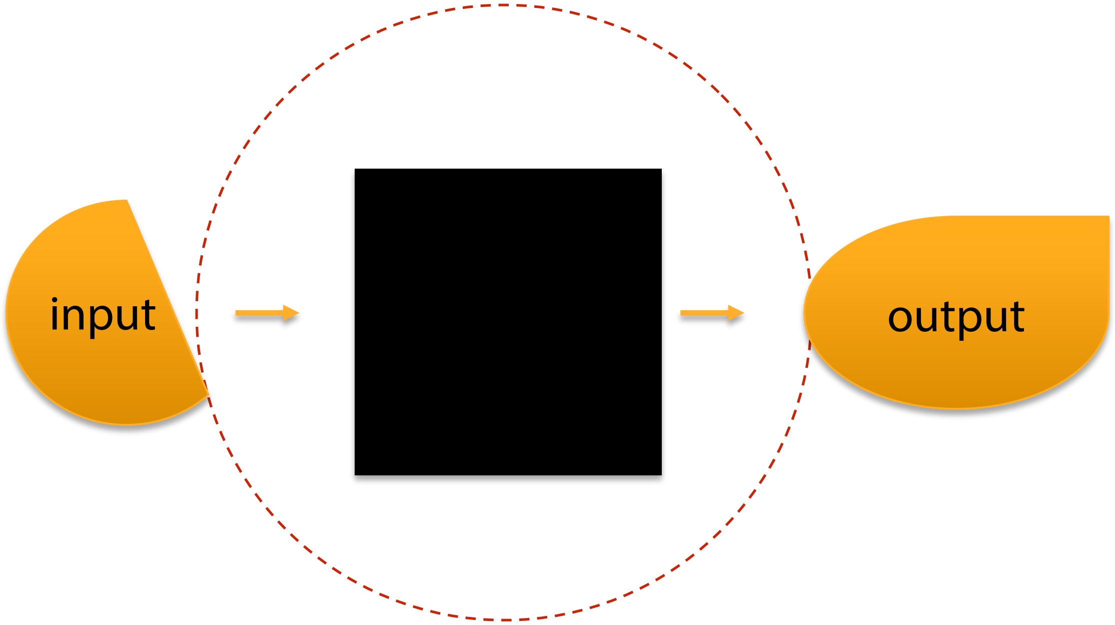
independent of any object instance

🚫 **access global state**

🚫 **modify input**

🚫 **change the world**


input
output



testable!

predictable!

🚫 access global state

🚫 modify input

🚫 change the world

Data In, Data Out

🚫 **surprises**

Data In, Data Out

- static methods
- @FunctionalInterface

☹		☺	
🚫	**type safety**	🚫	**hierarch y**
🚫	**propertie s**	○	**simple r**
○	**duplicatio n**	○	**easy to add**

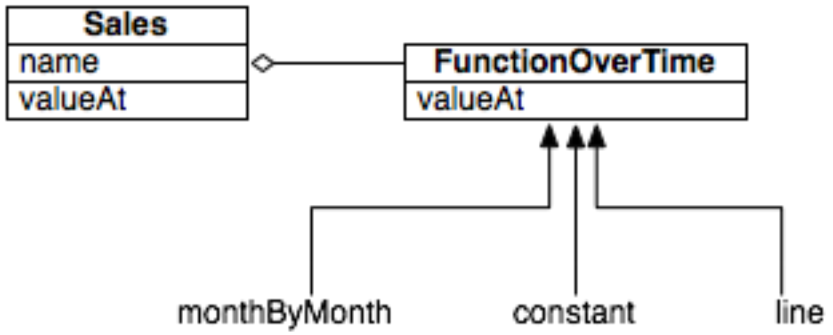
duplicatio

n

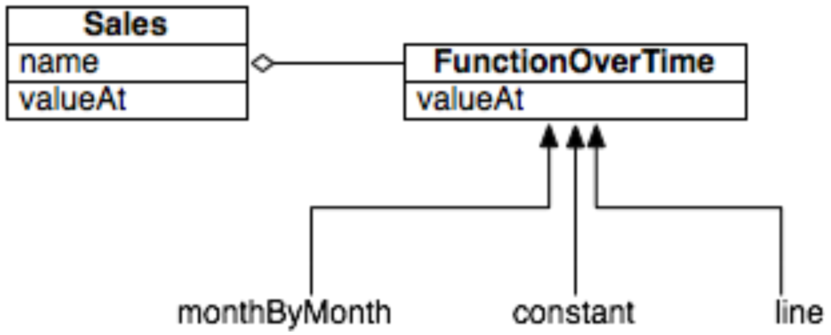
○	clear naming	🚫	duplicatio n
○	type safety	🚫	deep inheritance

Strateg

y


Sales
name
valueAt
FunctionOverTime
valueAt
monthByMonth
constant
line

Strateg
y

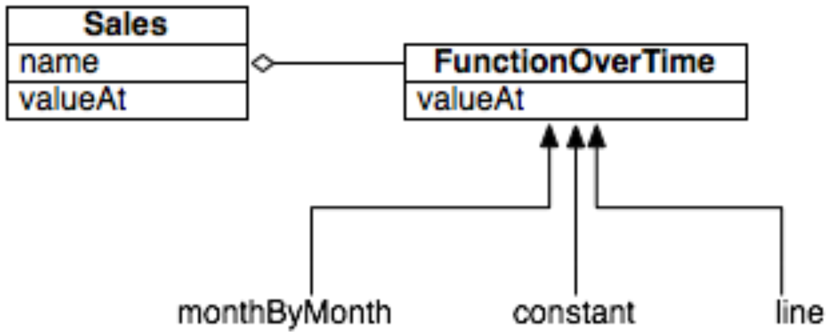


```
public interface FunctionOverTime {
  public Double valueAt(final Integer time);
}
```

```
new FunctionOverTime() {
  @Override
  public Double valueAt(final Integer time) {
    return array[time - 1];
  }
}
```

Strateg

y



Function<Integer, Double>

```
(time) -> array[time - 1];
```

You’ve learned a lot today.

Functions as Values

STORE functions in variables

PASS functions in parameters

RETURN functions from other functions

Functions as Values

(also known as)

First-class Functions

Higher-order Programming

Lambda Expression

```
(time) -> array[time - 1];
```

λ

Lambda

Lambdatron

Lambda Expression

```
(time) -> array[time - 1];
```

λ

Lambda

can see nearby variables

cannot change them!

~~Closure~~

can change nearby variables

Data In, Data Out

🚫 access global state

🚫 modify input

🚫 change the world

Data In, Data Out
(also known as)

Referential Transparency

```
FunctionOverTime c = (time) ->     ;          15.0

     "The result is " +             c(100)
```

Data In, Data Out

(also known as)

Referential Transparency

```
"The result is " +                15.0
```

Data In, Data Out
(also known as)

Referential Transparency

Pure Functions

🚫 **side effects**

Data In, Data Out

Functions

Data In, Data Out

Programs

Functional Programming:

Meaning

think about evaluation before execution

Mechanism

treat functions as values

- readabl

  e
- reliabl

  e

Fast!

Easy!

Java 8!