

**SE 2AA4, CS 2ME3 (Introduction to Software  
Development)**

**Winter 2017**

# **17 Functional Programming Continued**

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 10, 2017



# Functional Programming in Python

- Administrative details
- Map
- Anonymous functions
- Partial functions
- Filter
- Reduce (Fold)

# Administrative Details

- Notetaker needed for SE 2AA4
- Assignment 2
  - ▶ Files due by 11:59 pm Feb 19
  - ▶ File automatically sent to partners on Feb 20
  - ▶ Lab report due by 11:59 pm Feb 27
  - ▶ When returning an object, you can either create a new object or return a reference
  - ▶ `totalArea` is just the sum of the area of all circles, do not worry about overlap
  - ▶ Removed POS exception from syntax
  - ▶ Implementation does NOT have to look like the spec
- Midterm exam
  - ▶ March 1, 7:00 pm, TSH/120
  - ▶ 90 minute duration
  - ▶ Multiple choice - 30–40 questions?
  - ▶ Open book (any paper)

# Map

- Examples from [Learn You a Haskell for Great Good](#)
- Mathematical model:
  - ▶  $\text{map} : (a \rightarrow b) \times \text{seq of } a \rightarrow \text{seq of } b$
  - ▶  $\text{map} : (a \rightarrow b) \times [a] \rightarrow [b]$
- Python code: `map(func, seq)`

# Map Example

```
def add3(x):  
    return x + 3
```

```
print map(add3, [1, 5, 3, 1, 6])
```

- What do you think will be printed?
- What is the type of `add3`?
- What type does `map` return in this case?

# Anonymous Function Example

```
print map(lambda x: x + 3, [1, 5, 3, 1, 6])
```

or

```
add3 = lambda x: x + 3  
print map(add3, [1, 5, 3, 1, 6])
```

- lambda followed by list of arguments: expression
- Write code to add '!' to every string in a list of strings

# Introduce Partial Functions

- Write a function `repit(xs)` that takes a list `xs` and produces a new list that replicates each entry 3 times
  - ▶  $[1, 2, 3] \rightarrow [1, 1, 1], [2, 2, 2], [3, 3, 3]$
  - ▶  $\text{map} : (a \rightarrow b) \times [a] \rightarrow [b]$
  - ▶  $\text{map} : (t \rightarrow [t]) \times [t] \rightarrow [[t]]$
  - ▶  $\text{rep} : t \times \text{int} \rightarrow [t]$
- What if we could partially evaluate `rep`?

# Partial Functions

```
from functools import partial

def power(base, exponent):
    return base ** exponent

square = partial(power, exponent=2)
cube = partial(power, exponent=3)
```

Example from [Python Partials are Fun!](#)



# Partial Functions Example

```
from functools import partial
def rep(x, n):
    return [x for i in range(n)]
def repit(xs):
    rep3 = ?
    return map(rep3, xs)
```

What should ? be?

## Another Example with Map

Write a map that takes a 2D list of numbers and returns a 2D list where all elements are squared.

```
map (mapSqr, [[1,2], [3, 4, 5, 6], [7, 8]])
```

- What is the type of `mapSqr`?
- Can you write a function of this type for `mapSqr`?

# Filter

- Examples from [Learn You a Haskell for Great Good](#)
- A filter is a function that takes a predicate and a list and returns the list of elements that satisfies the predicate
- Mathematical model:
- $\text{map} : (a \rightarrow \text{Bool}) \times [a] \rightarrow [a]$
- Python code: `filter(func, seq)`
- Example
  - ▶ `filter(lambda x: x>3, [1,5,3,2,1,6,4,3,2])`
  - ▶ `filter(lambda x: x==3, [1,5,3,2,1,6,4,3,2])`

## Filter Example

Write a filter that takes a list of numbers and returns only the even numbers.

# Reduce

- `reduce(func, seq, accum)`
- $\text{reduce} : (a \times a \rightarrow a) \times [a] \times a \rightarrow a$
- Takes a binary function and applies it to the first two elements
- Takes the result and uses it in the binary function along with the next element
- Repeats until completely reduced
- `accum` is the initial value for the accumulator
- How would you use `reduce` to calculate the sum of the numbers from 1 to 100?
- How about the product?

# Standard Deviation

Write a function `stdDev(A)` that takes a list of numbers `A` and returns:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{|A|-1} (A_i - \mu)^2}{|A|}}$$

# And Series

Write a function `forall(A)` that takes a list of boolean values `A` and returns  $\forall(i : \mathbb{N} | i \in [0..|A| - 1] : A_i)$

# Contains

Write a function `contains(x, xs)` that takes a value `x` and a list of values `xs` and returns  $\exists(y|y \in xs : y = x)$



# Index Set

Write a function `indexSet(x, B)` that takes a value `x` and a list of values `B` and returns a list of indices where  $B[i] = x$ .

As a first step, how would you say this mathematically?

# Index Set

Write a function `indexSet(x, B)` that takes a value `x` and a list of values `B` and returns a list of indices where `B[i] = x`.

$$\text{indexSet}(x, B) \equiv \{i : \mathbb{N} \mid i \in [0..|B| - 1] \wedge B_i = x : i\}$$

How could you use `indexSet` to calculate `rank(x, A)`?