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

Winter 2018

16 Functional Programming Continued

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 9, 2018

16 Functional Programming Continued

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

Administrative Details

- Assignment 2
  - Part 1: February 20, 2018
  - Partner Files: February 26, 2018
  - Part 2: March 2, 2018
  - For floats remember to use pytests approx
  - Consider using interp1d - the implementation does not have to follow the spec, it just has to match the specified behaviour
  - A2 demo
- Midterm exam
  - Wednesday, February 28, 7:00 pm
  - 90 minute duration
  - Next week's tutorial will cover a sample midterm

HW: Remove Everything but Uppercase

Write a function `removeNonUpperCase(st)` that takes a string `st` and returns the string that results by removing all non upper case letters

How would you build the sequence of `['A','B',...,'Z']`?

```
def removeNonUpperCase(st):
    return [c for c in st if (c in upCase)]
```

HW: Remove Everything but Uppercase

Write a function `removeNonUpperCase(st)` that takes a string `st` and returns the string that results by removing all non upper case letters

How would you build the sequence of `['A','B',...,'Z']`?
`[chr(i) for i in range(?, ?)]`

```
def removeNonUpperCase(st):
    return [c for c in st if (c in upCase)]
```

HW: Remove Everything but Uppercase

Write a function `removeNonUpperCase(st)` that takes a string `st` and returns the string that results by removing all non upper case letters

How would you build the sequence of `['A','B',...,'Z']`?

```
[chr(i) for i in range(?, ?)]
[chr(i) for i in range(ord('A'),ord('Z')+1)]
```

```
def removeNonUpperCase(st):
    return [c for c in st if (c in upCase)]
```

Map

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

Map Example

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

list(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

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

or

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

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

```
list(map(?, ["BIFF","BANG","POW"]))
```

Anonymous Function Example

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

or

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

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

```
list(map(lambda s:  s+'!',
["BIFF","BANG","POW"]))
```

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]]`
  - map : $(a \rightarrow b) \times [a] \rightarrow [b]$
  - map : $(t \rightarrow [t]) \times [t] \rightarrow [[t]]$
  - rep : $t \times int \rightarrow [t]$
- What if we could partially evaluate rep?

```
def rep(x, n):
    return [x for i in range(n)]
```

Partial Functions

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

square = lambda x: power(x, 2)
cube = lambda x: power(x, 3)
```

Example from Python Partials are Fun!

Partial Functions Example

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

What should ? be?

Partial Functions Example

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

What should ? be?

```
rep3 = lambda x:  rep(x, 3)
```

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`?

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`?
  map : $(a \rightarrow b) \times [a] \rightarrow [b]$

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

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`?
  map : $(a \rightarrow b) \times [a] \rightarrow [b]$
  map : $(a \rightarrow b) \times [[int]] \rightarrow [[int]]$

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

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`?
  map : $(a \rightarrow b) \times [a] \rightarrow [b]$
  map : $(a \rightarrow b) \times [[int]] \rightarrow [[int]]$
  map : $([int] \rightarrow [int]) \times [[int]] \rightarrow [[int]]$

- 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:
- map : $(a \rightarrow Bool) \times [a] \rightarrow [a]$
- 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 from 1 to 10 and returns only the even numbers.

```
def even(x):
    return x%2==0

filter(func, seq)
```

Filter Example

Write a filter that takes a list of numbers from 1 to 10 and returns only the even numbers.

```
def even(x):
    return x%2==0

filter(func, seq)
filter(even, range(1, 11))
```

Reduce

- `reduce(func, seq, accum)`
- 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
- In Python 3 need `from functools import reduce`
- How would you use reduce to calculate the sum of the numbers from 1 to 5?
- How about the product?

Understanding Reduce for Sum

- `reduce(lambda x, y:  x+y, range(1, 6), 0)`
- `reduce(lambda x, y:  x+y, [1, 2, 3, 4, 5], 0)`
- `((((0+1)+2)+3)+4)+5`

Standard Deviation

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

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

```
from math import sqrt
from functools import reduce
def stdDev(A):
    add = lambda x, y: x+y
    N = float(len(A))
    mu = reduce(add, A, 0)/N
    tot = reduce(add, [(a - mu)**2 for a in A], 0)
    return sqrt(tot/N)

print(stdDev([13, 23, 12, 44, 55]))
```

And Series

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

```
def forall(A):
    return reduce(?, A, ?)

print(forall([True, True, True, True]))
print(forall([True, True, True, False]))
```

And Series

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

```
def forall(A):
    return reduce(?, A, ?)

print(forall([True, True, True, True]))
print(forall([True, True, True, False]))

reduce(lambda x, y:  x and y, A, True)
```

Contains

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

```
def contains(x, xs):
    return reduce(func?, seq?, False)

adjectives = ["lazy", "grouchy", "scheming"]

print(contains(4, [3, 4, 5, 6, 8]))
print(contains("happy", adjectives))
```

Contains

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

```
def contains(x, xs):
    return reduce(func?, seq?, False)

adjectives = ["lazy", "grouchy", "scheming"]

print(contains(4, [3, 4, 5, 6, 8]))
print(contains("happy", adjectives))


reduce(lambda a, b:  a or b, [y==x for y in xs],
False)
```

Connection Between FP Topics

- List comprehensions can be implemented by maps and filters
- All list comprehensions can be implemented via for loops
- Not all for loops can be implemented by a list comprehension
- Details here

```
N = range(10)
dbl = lambda n: n*2
odd = lambda n: n%2==1
list(map(dbl, filter(odd, N)))
# versus
[n * 2 for n in N if n % 2 == 1]
```

Homework: 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?

Homework: 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?

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

Homework: 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?

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

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