

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

**Winter 2017**

# **16 Functional Programming in Python**

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 10, 2017



# Functional Programming in Python

- Administrative details
- Functional programming
- Functional programming in Python
  - ▶ Defining functions
  - ▶ List comprehension
  - ▶ Map
  - ▶ Filter
  - ▶ Fold (Reduce)
  - ▶ Anonymous functions
  - ▶ Partial functions

# 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
- Midterm exam
  - ▶ March 1, 7:00 pm, TSH/120
  - ▶ 90 minute duration
  - ▶ Multiple choice - 30–40 questions?
  - ▶ Open book (any paper)

# Functional Programming

- Computation is treated as the evaluation of mathematical functions (not CS subroutines)
  - ▶ No state
  - ▶ No mutable data
  - ▶ Programming with expressions, not statements
- No side effects
- Easier to reason about than imperative or OO code
- Functions are a first order data type
  - ▶ Can pass functions as arguments
  - ▶ Can return functions
- Origin with lambda calculus
- There is a focus on list processing

# Functional Programming in Python

- Python was an imperative/OO language first
- Other functional languages (like Haskell) have pattern matching
- Python is dynamically typed
- Cannot inspect the type of a function

# List Comprehension

- Examples from [Learn You a Haskell for Great Good](#)
- Mathematical model:  $\langle E|R, P \rangle$ 
  - ▶  $E$  is an expression
  - ▶  $R$  is a sequence
  - ▶  $P$  is a predicate
- Python code: `E for dummy in R if P`
- Examples
  - ▶ `[x*2 for x in range(1, 11)]`
  - ▶ `[x*2 for x in range(1, 11) if x*2 >= 12]`
  - ▶ `radii = [c.radius() for c in circles]` from [A1](#)
  - ▶ [What output should be produced?](#)

# List Comprehension to find List Length

```
def length(xs):  
    return sum([? for x in xs])
```

What should ? be to return the length of xs?

# List Comprehension Examples Cont'd

- Write a function `rep(x, n)` that returns a list of `n` elements, where each element is `x`
- Write a function that takes a list of integers and replaces each odd number greater than 10 with "BANG!" and each odd number that's less than 10 with "BOOM!"
  - ▶ What is the expression for the list comprehension?
  - ▶ Do you know conditional expressions in Python?
  - ▶ `x = true_value if condition else false_value`
  - ▶ Can you write a Python function for odd?



# List Comprehension Two Lists

- Given the lists
  - ▶ `nouns = ["hobo", "frog", "pope"]`
  - ▶ `adjectives = ["lazy", "grouchy", "scheming"]`
- Write a list comprehension that concatenates all the adjectives with all the nouns

# 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

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

# Nested List Comprehension

Given a list of several lists of numbers, remove all odd numbers without flattening the list.

```
xxs =  
[[1,3,5,2,3,1,2,4,5], [1,2,3,4,5,6,7,8,9], [1,2,...
```

# 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)`
- $\text{reduce} : (a \times a \rightarrow a) \times [a] \rightarrow a$
- 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
- How would you use reduce to calculate the sum of the numbers from 1 to 100?
- How about the product?