Programming Languages
Functional languages intro

2020

Instructor: Odelia Schwartz

Introduction

- Zoom intros …
- Questions about assignment?

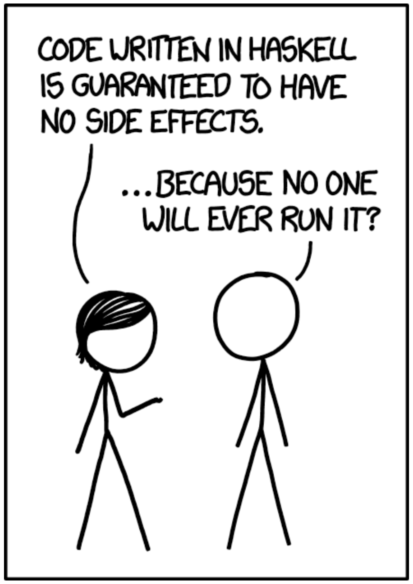
- Start with zoom introductions …
- Mute microphone unless asking questions
- Turn video off if too slow
- Give me feedback! (email / in class)

Introduction

- Imperative: based on Von Neumann
- Functional: based on mathematical functions

Introduction

- Imperative: based on Von Neumann
- Functional: based on mathematical functions
- Important feature of functional: no side effects; no variables; no states



Introduction

- Last decade: increase in interest and use of functional languages. What languages?

Introduction

- Last decade: increase in interest and use of functional languages. What languages?

  ML
  Haskell
  F#
  Scheme / Lisp
  Clojure

Introduction

- Last decade: increase in interest and use of functional languages. What languages?

  ML
  Haskell
  F#
  Scheme / Lisp
  Clojure

We'll largely focus on Scheme

Introduction

- Last decade: increase in interest and use of functional languages. What languages?

ML
Haskell
F#
Scheme / Lisp
Clojure

And some ML / Haskell

Introduction

- Last decade: increase in interest and use of functional languages. What languages?

  ML
  Haskell
  F#
  Scheme / Lisp
  Clojure

Functional capabilities also common in modern imperative languages!

Mathematical functions

Domain set → Range set

"

Mathematical functions

Domain set $\longrightarrow$ Range set

- Evaluation order of mapping expressions controlled by recursion and conditional expressions
- Since no side effects cannot depend on any external values; always map a particular element of domain to same element of range

Mathematical functions

Domain set ⟶ Range set

- Evaluation order of mapping expressions controlled by recursion and conditional expressions
- Since no side effects cannot depend on any external values; always map a particular element of domain to same element of range

Mathematical functions

Imperative in contrast:

- Subprograms may depend on current value of nonlocal or global variables…
- Difficult to determine statistically what values subprogram will produce due to side effects…

Simple Functions

Note: we are discussing math concepts that apply to PL; not yet PL …

Example:

$$cube(x) = x * x * x$$

Simple Functions

Note: we are discussing math concepts that apply to PL; not yet PL …

Example:

$$cube(x) = x * x * x$$

- Domain and range real numbers

Simple Functions

Note: we are discussing math concepts that apply to PL; not yet PL …

Example:

$$cube(x) = x * x * x$$

- Parameter x is fixed during evaluation (bound to a value from domain set)

$$cube(2.0) = 2.0 * 2.0 * 2.0 = 8.0$$

Lambda expressions

- Early theoretical work separated task of *defining* a function from that of *naming* a function

Lambda expressions

- Lambda notation (Church, 1941) provides method for defining nameless functions

Example:

$$\lambda(x)\, x * x * x$$

Lambda expressions

- Lambda notation (Church, 1941) provides method for defining nameless functions

Example:

$$(\lambda(x)\, x * x * x)\ (2)$$

Evaluates to?

Lambda expressions

- Lambda notation (Church, 1941) provides method for defining nameless functions

Example:

$$(\lambda(x)\, x * x * x)\ (2)$$

Evaluates to? 8

Lambda expressions

- Python example

open google colab or jupyter notebook

```
x = lambda a: a * a * a

print(x(5))
```

Lambda expressions

- Lambda notation (Church, 1941) provides method for defining nameless functions

- Church defined formal system for function definition, function application, and recursion using lambda functions (*lambda calculus*)

- Inspiration for functional languages

Functional forms

- Higher order functions or functional form: takes one or more functions as parameters, or yields a function as a result, or both

Functional forms

- Common type: **functional composition**

$$h = f \; o \; g$$

Means:

$$h = f(g(x))$$

Functional forms

- Common type: **functional composition**

$h = f \ o \ g$

Means:

$h = f(g(x))$

Example:

$f(x) = x + 2$

$g(x) = 3 * x$

$h = f\big(g(x)\big) =$

Functional forms

- Common type: **functional composition**

$$h = f \; o \; g$$

Means:

$$h = f(g(x))$$

Example:

$$f(x) = x + 2$$
$$g(x) = 3 * x$$

$$h = f\big(g(x)\big) = 3 * x + 2$$

Functional forms

- Common type: **apply to all**
  (often called *map* in PL)

  Functional form that takes a single function as a parameter. Applies function to each of the values in a list, returning a list

Functional forms

- Common type: **apply to all**
  (often called *map* in PL)

  Functional form that takes a single function as a parameter. Applies function to each of the values in a list, returning a list (math symbol α)

Functional forms

- Common type: **apply to all**
  (often called *map* in PL)

  Functional form that takes a single function as a parameter. Applies function to each of the values in a list, returning a list (math symbol α)

Example:

$$h(x) = x * x$$

$$\alpha\big(h, (2,3,4)\big) =?$$

Functional forms

- Common type: **apply to all**
  (often called *map* in PL)

  Functional form that takes a single function as a parameter. Applies function to each of the values in a list, returning a list (math symbol α)

Example:

$$h(x) = x * x$$

$$\alpha\big(h, (2,3,4)\big) = (4, 9, 16)$$

Lambda expressions

- Python example

  open google colab or jupyter notebook

```
# http://book.pythontips.com/en/latest/map_filter.html

items = [1, 2, 3, 4, 5]

cubed = list(map(lambda x: x**3, items))
```

Lambda expressions

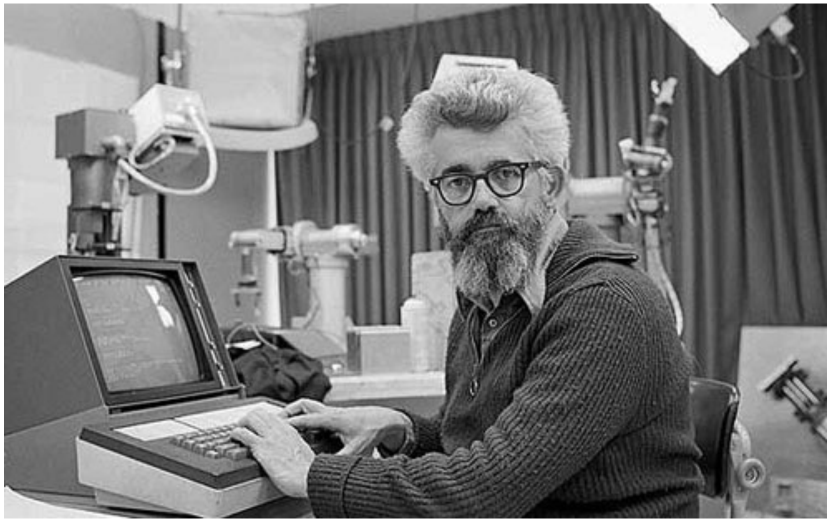
- Python example

Compare to:

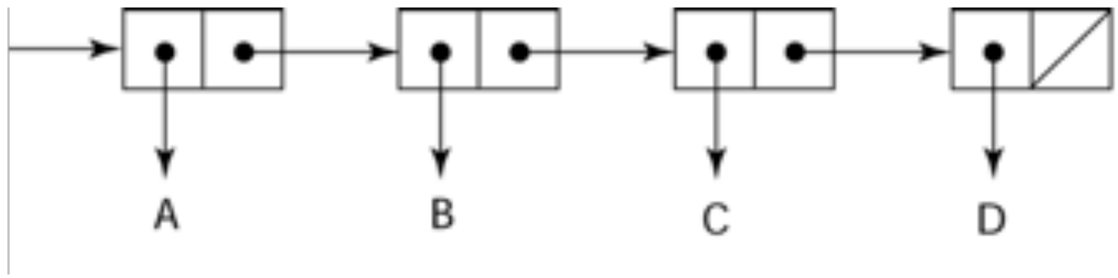
```
items = [1, 2, 3, 4, 5]
cubed = []
for i in items:
   cubed.append(i**3)
print(cubed)
```

Lisp

- McCarthy, MIT, 1959
- Functional through Lisp like imperative through Fortran: first language but no longer represents latest design concepts
- Scheme, which we will learn in detail, has similarities

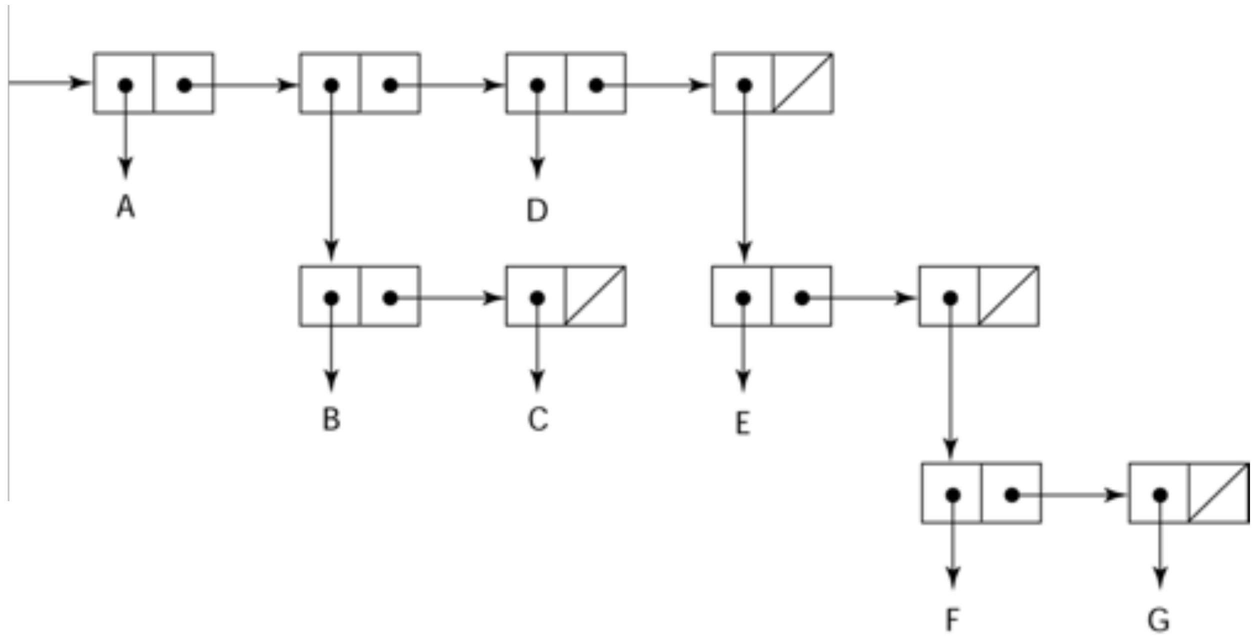


Lisp



- Representing list (A B C D)
- Internal representation as linked lists

Lisp



and `(A (B C) D (E (F G)))`

Lisp

- List (A B C)
- If interpreted as data, it’s a simple list of 3 elements: A, B, C

Lisp

- List (A B C)

- If interpreted as data, it's a simple list of 3 elements: A, B, C

- If interpreted as a function, it means that function A is applied to two parameters: B and C

Lisp

- List (A B C)
- If interpreted as data, it's a simple list of 3 elements: A, B, C
- If interpreted as a function, it means that function A is applied to two parameters: B and C

(in a sense, no separation of data and code…)

Lisp

- List (A B C)
- If interpreted as data, it's a simple list of 3 elements: A, B, C
- If interpreted as a function, it means that function A is applied to two parameters: B and C

Example: (+ 5 7) evaluates to 12

Lisp

- Lambda notation chosen to specify function definition, but modified to also allow binding of functions to names

(function_name (LAMBDA (param1 .. Param n) expression))

Lisp

- Lambda notation chosen to specify function definition, but modified to also allow binding of functions to names

(function_name (LAMBDA (param1 .. Param n) expression))

Why sometimes no need for a function name?

Lisp

- Lambda notation chosen to specify function definition, but modified to also allow binding of functions to names

```
(function_name (LAMBDA (param1 .. Param n) expression ))
```

Why sometimes no need for a function name?

Example: function for immediate application to a parameter list; produced function has no need for a name, since applied only at one point in construction

Next class

- Next class Scheme; more in depth

Using Scheme interpreter

- Next class Scheme; more in depth
- We will run code using Chicken Scheme
- Installing on your computer:

  https://wiki.call-cc.org/platforms

- Can also run online with different interpreter, works on simple examples I have tested: https://repl.it/languages/scheme

Using Scheme interpreter

Using Chicken Scheme:

- Type csi in the terminal. It will open the chicken interpreter.
- ,q to quit
- Chicken interpreter uses lower case for reserved words (book and some other interpreters use upper case)

Using Scheme interpreter

Our department computer also has Chicken Scheme:

- Log onto Johnston
- Then log onto one of the computers, such as wilderness etc.
- Type csi in the terminal. It will open the chicken interpreter