Lambdas
Learning Objectives

• Lambda are anonymous functions, which use expressions
  - We don’t use return, they always return the value.
  - They are typically short and concise
  - They don’t have an “intrinsic” name when using an environment diagram.
• Function expression
  - “anonymous” function creation
  - Expression, not a statement, no return or any other statement

\[
\text{lambda <arg or arg\_tuple> : <expression using args>}
\]

```python
inc = lambda v : v + 1
def inc(v):
    return v + 1
```
Lambdas

```python
>>> def inc_maker(i):
...     return lambda x: x + i
...

>>> inc_maker(3)
<function inc_maker.<locals>.<lambda> at 0x10073c510>

>>> inc_maker(3)(4)
7

>>> map(lambda x: x*x, [1, 2, 3, 4])
<map object at 0x1020950b8>

>>> list(map(lambda x: x*x, [1, 2, 3, 4]))
[1, 4, 9, 16]
```

UC Berkeley | Computer Science 88 | Michael Ball | http://cs88.org
Lambda with HOFs

- A function that returns (makes) a function

```python
def leq_maker(c):
    return lambda val: val <= c

>>> leq_maker(3)
<function leq_maker.<locals>.<lambda> at 0x1019d8c80>

>>> leq_maker(3)(4)
False

>>> filter(leq_maker(3), [0,1,2,3,4,5,6,7])
[0, 1, 2, 3]
```
Types and Universality
Learning Objectives

• Abstract Data Type:
  - Applying a name to a new concept.
  - Give us structure for organizing how we build and name functions
  - Write a simple ADT around a “point”.
Universality

• Everything that can be computed, can be computed with what you know now.
• “Turning Completeness”
• Poorly or Well
Evolution of Programming Languages

Mother Tongues
Tracing the roots of computer languages through the ages

Survival of the Fittest
Reasons a language endures, with examples of some classic tongues

Appals to a wide audience C (bestowed by the popularity of Unix)
Gets a job done Cobol (designed for business-report writing)
Delivers new functionality Java (runs on any hardware platform)
Fills a niche Mathematica (speaks up complex computations)
Offers a medium of elegance Icon (too friendly, line-oriented syntax)
Has a powerful user base or leader C++ (developed by Microsoft for .Net)
Has a charismatic leader Perl (programmer-author Larry Wall)

Just like half of the world's spoken tongues, most of the 3,388-plus computer programming languages are either endangered or extinct. As unwieldyness C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of O2.

Code-namer Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to resurrect and, in some cases, modernize languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure - here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at: HTTP://www.informatics.stanford.edu/units/long_list.html - Michael Mandan

Source: Paul Steen; Brent Hedges, associate director of computer science at BBN Research; The Reincorporating Museum; Todd Prentki, senior researcher at Microsoft; Gia Medharden, computer scientist, Stanford University.
Examples Data Types You have seen

• Lists
  – Constructors:
    » list(...)
    » [ <exps>, ... ]
    » [<exp> for <var> in <list> [ if <exp> ] ]
  – Selectors: <list> [ <index or slice> ]
  – Operations: in, not in, +, *, len, min, max
    » Mutable ones too (but not yet)

• Tuples. [We have only briefly used these.]
  – Constructors:
    » ( <exps>, ... )
  – Selectors: <tuple> [ <index or slice> ]
  – Operations: in, not in, +, *, len, min, max
More “Built-in” Examples

• Strings
  - Constructors:
    » `str( ... )`
    » "<chars>", ‘<chars>’
  - Selectors: `<str> [ <index or slice> ]`
  - Operations: in, not in, +, *, len, min, max

• Range
  - Constructors:
    » `range(<end>), range(<start>,<end>), range(<start>,<end>,<step>)`
  - Selectors: `<range> [ <index or slice> ]`
  - Operations: in, not in, len, min, max
Abstract Data Type

• We have “built-in” types, but we can use functions to represent some similar kind of data.
• Today we’re going to look at a “point”
• Functions are lightweight. Later we will build more complex structures.
Abstract Data Type

Operations

Object

Constructors
Selectors
Operations
External Representation

A new Data Type

Internal Representation

Implementation on that Internal representation

UC Berkeley | Computer Science 88 | Michael Ball | http://cs88.org
Example: A “Point”

• A point (in Geometry) is typically an x-y pair. (Though, we could define another interface...)

• Constructor: point()
• Selectors: x(), y()
• Operations: distance()
• Implementation: Could be anything! This uses a tuple.

• Python Tutor
Dictionaries
Learning Objectives

• Dictionaries store key-value pairs
  - ”a” maps to some value “b”
  - Each key in each dictionary maps to one value.
  - { ‘a’ , 1, ‘b’: 2 ... } 
• You can change the values of the dictionary
• You can change the keys, including adding and removing
Dictionaries are Mutable, too

• Immutable – the value of the object cannot be changed
  – integers, floats, booleans
  – strings, tuples
• Mutable – the value of the object can change
  – Lists
    – Dictionaries

```
>>> alist = [1, 2, 3, 4]
>>> alist
[1, 2, 3, 4]
>>> alist[2]
3
>>> alist[2] = 'elephant'
>>> alist
[1, 2, 'elephant', 4]

>>> adict = {'a': 1, 'b': 2}
>>> adict
{'b': 2, 'a': 1}
>>> adict['b']
2
>>> adict['b'] = 42
>>> adict['c'] = 'elephant'
>>> adict
{'b': 42, 'c': 'elephant', 'a': 1}
```
Dictionaries – by example

Constructors:
- `dict( hi=32, lo=17)`
- `dict([(hi,212),(lo,32),(17,3)])`
- `{x:1, y:2, 3:4}`
- `{wd:len(wd) for wd in "The quick brown fox".split()}`

Selectors:
- `water['lo']`
- `<dict>.keys(), .items(), .values()`
- `<dict>.get(key [, default] )`

Operations:
- `in, not in, len, min, max`
- `'lo' in water`

Mutators:
- `water['lo' ] = 33`