

Computational Structures in Data Science

Lambdas
Environments
Dictionaries



Announcements

- Reminder: Check Ed for links/forms for Extensions and switching for CS61A

Computational Structures in Data Science

Lambda Expressions



Learning Objectives

- Lambda are anonymous functions, which are expressions
 - Don't use `return`, lambdas always return the value of the expression.
 - They are typically short and concise
 - They don't have an "intrinsic" name when using an environment diagram.
 - Their name is the character λ

Why Use `λ`ambda?

- We often can use the behavior of simple function!
- Using functions gives us flexibility
- "Inline" functions are faster/easier to write, and sometimes require less reading.
- They're not "reusable", but that's OK!

λambda

Function expression

“anonymous” function creation

```
lambda <arg or arg_tuple> : <expression using args>
```

Expression, not a statement, no return or any other statement

```
add_one = lambda v : v + 1
```

```
def add_one(v):  
    return v + 1
```

Examples

```
>>> def make_adder(i):  
...     return lambda x: x+i  
...  
>>> make_adder(3)  
<function make_adder.<locals>.<lambda> at  
0x10073c510>  
  
>>> make_adder(3)(4)  
7  
  
>>> list(map(make_adder(3), [1, 2, 3, 4]))  
[4, 5, 6, 7]  
>>>
```

Lambda with HOFs

- **A function that returns (makes) a function**

```
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]
```


Sorting Data

- It is often useful to sort data.
- What property should we sort on?
 - Numbers: We can clearly sort.
 - What about the length of a word?
 - Alphabetically?
 - What about sorting a complex data set, but 1 attribute?
 - Imagine I have a list of courses: I could sort by course name, number of units, start time, etc.
- Python provides 1 function which allows us to provide a *lambda* to control its behavior

More Python HOFs

- sorted – sorts a list of data
- min
- max

All three take in an optional argument called `key` which allows us to control how the function performs its action. They are more similar to `filter` than `map`.

```
max([1,2,3,4,5], key = lambda x: -x)
```

`key` is the name of the argument and a `lambda` is its value.

```
fruits = ["pear", "grape", "KIWI", "APPLE",  
"melon", "ORANGE", "BANANA"]
```

```
sorted(key=lambda x: x.lower())
```

Sorting with Lambdas

```
>>> sorted([1,2,3,4,5], key = lambda x: x)
[1, 2, 3, 4, 5]
>>> sorted([1,2,3,4,5], key = lambda x: -x)
[5, 4, 3, 2, 1]
# Nonsensical pairing of numbers and words...
>>> sorted([(2, "hi"), (1, "how"), (5, "goes"), (7, "it")],
           key = lambda x:x[0])
[(1, 'how'), (2, 'hi'), (5, 'goes'), (7, 'it')]
>>> sorted([(2, "hi"), (1, "how"), (5, "goes"), (7, "it")],
           key = lambda x:x[1])
[(7, 'it'), (5, 'goes'), (2, 'hi'), (1, 'how')]
>>> sorted([(2,"hi"),(1,"how"),(5,"goes"),(7,"it")],
           key = lambda x: len(x[1]))
[(7, 'it'), (2, 'hi'), (1, 'how'), (5, 'goes')]
```

Computational Structures in Data Science

Environment Diagrams



Revisiting Environments

```
def make_adder(n):  
    return lambda k: k + n
```

```
add_2 = make_adder(2)
```

```
add_3 = make_adder(3)
```

```
x = add_2(5)
```

```
y = add_3(x)
```

Revisiting Environments: compose w/lambda

```
def make_adder(n):  
    return lambda k: k + n  
def compose(f, g):  
    return lambda x: f(g(x))  
  
add_2 = make_adder(2)  
add_3 = make_adder(3)  
add_5 = compose(add_2, add_3)  
  
x = add_2(2)  
z = add_5(x)
```

Environment Diagrams

- Organizational tools that help you understand code
- **Terminology:**
 - **Frame:** keeps track of variable-to-value bindings, each function call has a frame
 - **Global Frame:** global for short, the starting frame of all python programs, doesn't correspond to a specific function
 - **Parent Frame:** The frame of where a function is defined (default parent frame is global)
 - **Frame number:** What we use to keep track of frames, f1, f2, f3, etc
 - **Variable vs Value:** $x = 1$. x is the **variable**, 1 is the **value**

Environment Diagrams Rules

1. Always draw the global frame first
2. When evaluating assignments (lines with single equal), always evaluate right side first
3. When you **CALL** a function MAKE A NEW FRAME!
4. When assigning a primitive expression (number, boolean, string) write the value in the box
5. When assigning anything else (lists, functions, etc.), draw an arrow to the value
6. When calling a function, name the frame with the intrinsic name – the name of the function that variable points to
7. The parent frame of a function is the frame in which it was defined in (default parent frame is global)
8. If the value for a variable doesn't exist in the current frame, search in the parent frame

Demo

Example 1:

- [make_adder Higher Order Function: Environment Diagram Python Tutor Link](#)

Example 2:

- [Compose Python Tutor Link](#)

Computational Structures in Data Science

Taking a Break



Computational Structures in Data Science

Dictionaries



Learning Objectives

- Dictionaries are a new type in Python
- Lists let us index a value by a number, or position.
- Dictionaries let us index data by other kinds of data.

Dictionaries

- Constructors:
 - `dict(<list of 2-tuples>)`
 - `dict(<key>=<val>, ...)` # like kwargs
 - `{ <key exp>:<val exp>, ... }`
 - `{ <key>:<val> for <iteration expression> }`
 - `>>> {x:y for x,y in zip(["a","b"],[1,2])}`
 - `{'a': 1, 'b': 2}`
- Selectors: `<dict>[<key>]`
 - `<dict>.keys(), .items(), .values()`
 - `<dict>.get(key [, default])`
- Operations:
 - Key in, not in, len, min, max
 - `<dict>[<key>] = <val>`

Example Dictionary Operations

```
In [1]: text = "Once upon a time"  
d = {word : len(word) for word in text.split()}  
d
```

```
Out[1]: {'Once': 4, 'a': 1, 'time': 4, 'upon': 4}
```

```
In [2]: d['Once']
```

```
Out[2]: 4
```

```
In [3]: d.items()
```

```
Out[3]: [('a', 1), ('time', 4), ('upon', 4), ('Once', 4)]
```

```
In [4]: for (k,v) in d.items():  
        print(k,"=>",v)
```

```
('a', '=>', 1)  
( 'time', '=>', 4)  
( 'upon', '=>', 4)  
( 'Once', '=>', 4)
```

```
In [5]: d.keys()
```

```
Out[5]: ['a', 'time', 'upon', 'Once']
```

```
In [6]: d.values()
```

```
Out[6]: [1, 4, 4, 4]
```