

DISCUSSION 02

Environment Diagrams, Higher-Order Functions

Mingxiao Wei

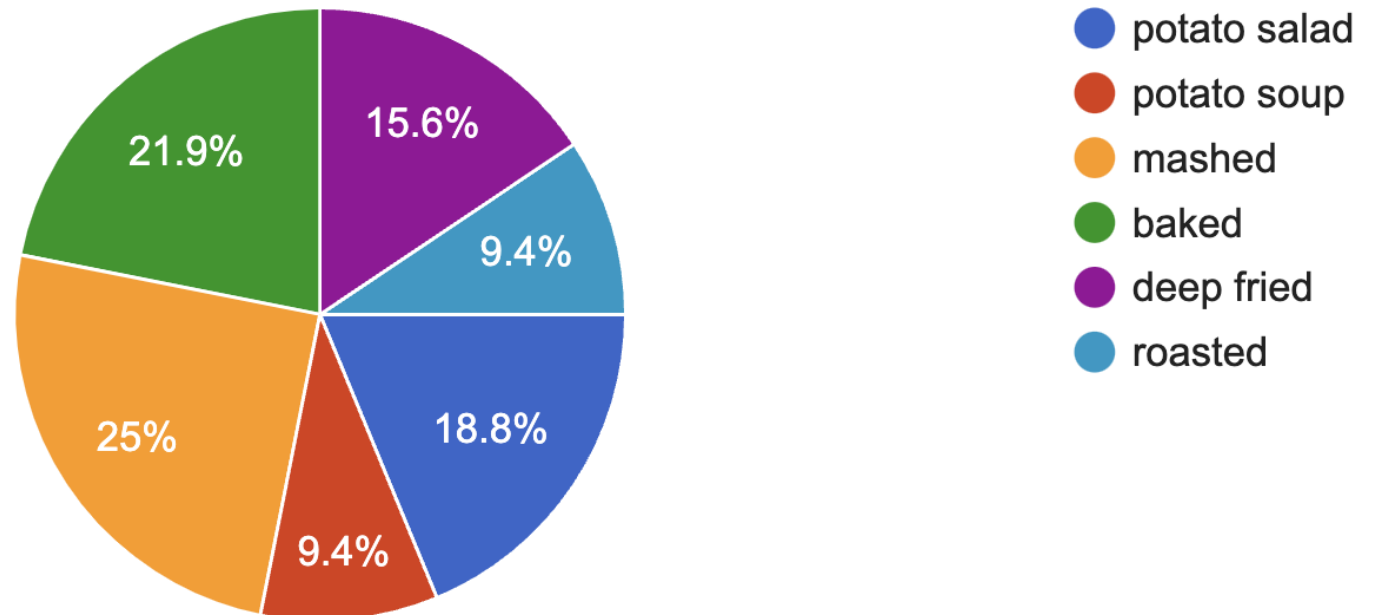
mingxiaowei@berkeley.edu

Feb 2, 2022


FROM LAST TIME... 👁️👁️

If you were a potato, how would you like to be cooked?

32 responses



LOGISTICS

- Homework 02 due today 02/02 @ 11:59pm
- Hog 
 - You can choose to work alone or in group of 2
 - If you work with a partner, only one of you need to submit and add the other on Gradescope!
 - Checkpoint 1 due next Tue 02/07
 - The whole project due next Fri 02/10
 - Submit the whole project by next Thu 02/09 for one extra point!
- Sign up for [tutoring sections](#) or [CSM sections](#)
 - Not required, but a good place to go if you want more practice!

ABOUT MIDTERM 1

Yes, it's coming ...

- **Default: Next Monday (02/06) 7-9pm, in-person, right-handed desk**
- If you need any type of accommodations (left-handed desk, remote exam, alternate time, extended time, etc.), please [fill out this form](#) by today!!
- Check out [Ed post #376](#) for more details!
- Pro Tips
 - Make sure you are (somewhat) comfortable with all the topics in scope
 - Familiarize yourself with the [study guide](#) beforehand
 - This was from last semester; the updated version will be linked in the [midterm logistics post](#) once it's ready
 - Do [past exams](#)
 - For each question, make sure you understand it before moving on to the next exam
 - Consult walkthrough videos! They are super helpful
 - Post on Ed [thread](#) / come to OH for help

HIGHER ORDER FUNCTIONS



HIGHER ORDER FUNCTIONS

- A **higher order function (HOF)** is a function that manipulates other functions by taking in functions as arguments, returning a function, or both.
- HOFs are powerful abstraction tools that allow us to express certain general patterns as named concepts in our programs.

```
1  def composer(func1, func2):  
2      """Return a function f, such that f(x) = func1(func2(x))."""  
3      def f(x):  
4          return func1(func2(x))  
5      return f
```

HIGHER ORDER FUNCTIONS

Python 3.6

```
→ 1 x = 4
   2 def add_num(x):
   3     return lambda y: x + y
   4
   5 add_two = add_num(2)
   6 add_two(3)
```

[Edit this code](#)

→ line that just executed

→ next line to execute

< Prev Next >

Step 1 of 10

Visualized with pythontutor.com

NEW: [subscribe](#) to our YouTube

[Move and hide objects](#)

Frames

Objects

WORKSHEET Q4

CURRYING



CURRYING

- Currying - converting a function that takes multiple arguments into a chain of functions that each take a single argument

```
1  def curried_pow(x):  
2      def h(y):  
3          return pow(x, y)  
4      return h  
5  curried_pow(2)(3) # same as pow(2, 3)
```

CURRYING

- Why is it useful?

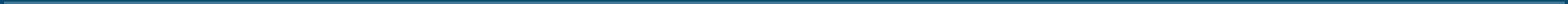
```
>>> pow(2, 3)
8
>>> pow(2, 4)
16
>>> pow(2, 10)
1024
```

```
>>> pow_2 = curried_pow(2)
>>> pow_2(3)
8
>>> pow_2(4)
16
>>> pow_2(10)
1024
```

- In contexts where only one-argument functions are allowed, such as with the `map` function, which applies a one-argument function to every term in a sequence, we can curry a multiple-argument function into a one-argument function

WORKSHEET Q6, 8, 7

CALL EXPRESSIONS



FUNCTION VALUES

- Values that represent functions, also known as function objects
- In general, there are 2 parts to a function:
 - The code (function body), represented by the intrinsic name
 - The parent frame - where the function is *defined*
- The function body is not evaluated until we call the function!
- To call a function, write the name of the variable that corresponds to the function value, write a parenthesis, and put all parameters for the function, if any - this is just a call expression

FUNCTION VALUES

- Intrinsic name

- For a function defined using the `def` statement, for example:

```
1  def some_func(x, y):  
2      return x * 2 + y * 5
```

Here `some_func` is the intrinsic name

- For a lambda function, its intrinsic name is just `λ`.
 - To differentiate between lambda functions, we often add the line number where the lambda is defined.
For example:

```
1  # global frame  
2  other_func = lambda x, y: x * 2 + y * 5
```

Here `other_func` will be represented as `λ(x, y) <line 2>` in the environment diagram

FUNCTION VALUES

```
def go(bears):  
    print(print(bears))  
    print('GO BEARS!')  
    return 'gob ears'
```

```
>>> go  
Function  
>>> go('bruh')  
bruh  
None  
GO BEARS!  
'gob ears'
```


EVALUATING A CALL EXPRESSION

1. Evaluate the operator, which should evaluate to a function value.
2. Evaluate the operands from left to right.
3. Draw a new frame, labelling it with the following:
 - A unique index (`f1`, `f2`, `f3`, ...)
 - The intrinsic name of the function, which is also the name of the function object itself
 - The parent frame (`[parent = ...]`)
4. Bind the formal parameters to the argument values obtained in step #2
5. Evaluate the body of the function in this new frame until a return value is obtained. Write down the return value in the frame.

* If a function does not have a return value, it implicitly returns `None`

* For built-in or imported functions like `abs` and `add`, we do not need to draw a new frame when calling them

WORKSHEET Q1-2

LAMBDA EXPRESSIONS



LAMBDA EXPRESSIONS

- A lambda expression evaluates to **function values** but does not bind it to a name, unless we bind it to some names using assignment statements
- Anatomy:

```
f = lambda <p1>, <p2>, ...: <returned expr>
```

This is equivalent to (though not the same as)

```
def f(<p1>, <p2>, ...):  
    return <returned expr>
```

- Similarly, the return expression of a lambda function is not evaluated until the lambda is called
- Unlike `def` statements, lambda expressions can be used as an operator or an operand to a call expression. This is because they are simply one-line expressions that **evaluate to functions**

LAMBDA EXPRESSIONS

	<code>lambda</code>	<code>def</code>
Type	Expression that evaluates to a value	Statement that alters the environment
Results of execution	Creates an anonymous <code>lambda</code> function with no intrinsic name.	Creates a function with an intrinsic name and binds it to that name in the current environment.
Effect on the environment	Does not create or modify any variables.	Both creates a new function object and binds it to a name in the current environment.
Usage	A lambda expression can be used anywhere that expects an expression, such as in an assignment statement or as the operator or operand to a call expression.	After executing a <code>def</code> statement, the created function is bound to a name. You should use this name to refer to the function anywhere that expects an expression.

LAMBDA EXPRESSIONS

```
>>> what = lambda x: x + 5
>>> what
<function <lambda> at ...>
>>> what(4)
9
>>> (lambda y: y + 5)(4)
9
>>> (lambda f, x: -f(x))(lambda y: y + 1, 10)
-11
```

WORKSHEET Q3

ATTENDANCE! 🤠

go.cs61a.org/mingxiao-att

- The attendance form and slides are both linked on our section website!
 - go.cs61a.org/mingxiao-index
- Please do remember to fill out the form by midnight today!!

