DISCUSSION 02

Environment Diagrams, Higher-Order Functions

Mingxiao Wei
mingxiaowei@berkeley.edu

Sep 8, 2022

LOGISTICS 🏡

- Homework 02 due today 09/08 @ 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 okpy!
  - The whole project due next Fri 09/09
  - Submit the whole project by today 09/08 for one extra point!
- First round of scores are out on howamidoing - if there's any discrepancy in your attendance for sections, please email me and include `CS 61A` in the title!
- Homework 01 Recovery - due next Mon 09/12, check out Ed post #576 for more details
- Section change deadline is next Wed 09/14
  - If you'd like to switch to other sections, check sections.cs61a.org for open ones!

ABOUT MIDTERM 1 🐿️

- **Default: Next Monday (09/12) 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 tomorrow 09/09!!
- Check out Ed post #667 for more details!
- Pro Tips
  - Make sure you are (somewhat) comfortable with all the topics in scope
  - Attend exam review sessions (mentioned in the post above) if you'd like more review by topic
  - Familiarize yourself with the study guide beforehand
  - Do past exams
    - Check out Ed post #678 to see which exam questions are in scope for this semester
    - 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

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:

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

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

```
# global frame
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

ENVIRONMENT DIAGRAMS 🛤️

EVALUATION RULES

Call Expressions

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 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

EVALUATION RULES

Assignment Statements

1. Evaluate all of the expressions on the RHS of the assignment operator (the single equal sign) from left to right
2. Bind all the names on the LHS to the resulting values in the current frame

* Names can only bind to values, not other names!

```
a, b = 1, 2
a, b = b, a
# In Python, you can swap the values of multiple variables in one line
a, b = b + 1, a + b
# If there are multiple arguments to print, they will be seperated
# by a whitespace in the outputted line
print("a =", a, ", b =", b)
# output: a = 3, b = 3
```

EVALUATION RULES

Variable Lookup

1. Look it up in the current frame
2. If not found, go look in the parent frame, and up and up until global. If it's not there, then error.

   * Built-in functions like `max` and `min` are usually not displayed on the environment diagram.

ENVIRONMENT DIAGRAMS

- An environment diagram keeps track of all the variables that have been defined and the values they are bound to.
- Frames
  - By default, we start with the global frame
  - A new frame is created whenever a function is called - though we don't need to draw the frames for built-in functions
  - For each frame (except for the global frame):
    - Write down the intrinsic name of the function and its parent frame
    - When the function returns, write the the return value for that frame
- Bindings
  - Keep track of the values corresponding to each variable
  - Created when there's an assignment statement/def statement or when calling a function - we need to bind its parameters to their values after opening a new frame for the call

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

Example of using lambda expressions directly as operators:

```
1  >>> (lambda y: y + 5)(4)
2  9
3  >>> (lambda f, x: f(x))(lambda y: y + 1, 10)
4  11
```

Bonus: draw the environment diagram when evaluating the second line!

WORKSHEET - Q3

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.

```
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
```

WORKSHEET - Q4,5,6

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 - Q7

ATTENDANCE! 🤠

go.cs61a.org/mingxiao-att

- The attendance form and slides are both linked on our section website!