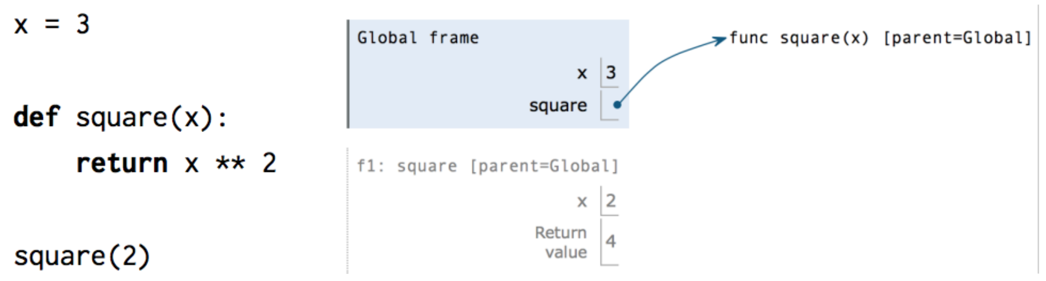
Python — Structures and Interpretations of Computer Programs

Winter 2017 — Hyun Jae Moon — Lecture 2

1 Environmental Diagrams

What is an Environmental Diagram?

An environment diagram keeps track of all the variables that have been defined and the values they are bound to.



```
x = 3


def square(x):
    return x ** 2


square(2)
```

When you execute assignment statements in an environment diagram (like x = 3), you need to record the variable name and the value:

1. Evaluate the expression on the right side of the = sign
2. Write the variable name and the expression's value in the current frame.

When you execute def statements, you need to record the function name and bind the function object to the name.

1. Write the function name (e.g., square) in the frame and point it to a function object (e.g., func square(x) [parent=Global]). The [parent=Global] denotes the frame in which the function was defined.

When you execute a call expression (like square(2)), you need to create a new frame to keep track of local variables.

1. Draw a new frame. Label it with
   - a unique index (f1, f2, f3 and so on)
   - the intrinsic name of the function (square), which is the name of the function object itself. For example, if the function object is func square(x) [parent=Global], the intrinsic name is square.
   - the parent frame ([parent=Global])
2. Bind the formal parameters to the arguments passed in (e.g. bind x to 3).
3. Evaluate the body of the function.

If a function does not have a return value, it implicitly returns None. Thus, the ?Return value? box should contain None.

Since we do not know how built-in functions like add(...) or min(...) are implemented, we do not draw a new frame when we call built-in functions.

Questions

1. Draw the environment diagram that results from running the following code.

```
a = 1
def b(b):
    return a + b
a = b(a)
a = b(a)
```

2. Draw the environment diagram so we can visualize exactly how Python evaluates the code. What is the output of running this code in the interpreter?

```
from operator import add
def sub(a, b):
    sub = add
    return a - b
add = sub
sub = min
print(add(2, sub(2, 3)))
```

2 More Functions

Higher Order Functions

A higher order function is a function that manipulates other functions by taking in functions as arguments, returning a function, or both. We will be exploring many applications of higher order functions. Let's take a look at an example.

```
>>> def f(x):
...     def g(y):
...         return x + y
...     return g
...
>>> add_one = f(1)
>>> add_one(2)
3
>>> add_two = f(2)
>>> three = add_two(1)
>>> three + 1
4
>>> f(3)(4)
7
```

Here, when we call the function f on 1, it would return the inner function g with an information that x is 1. If we were to call the returned function to another integer, it would add that integer to 1 and return that. The Important catch is that functions can be defined within functions, and can be utilized within it. Make sure to check what you are returning and where you are at. Keep in track of all the parameters and functions. If you are curious of the frames, use pythontutor online.

Question: Make a Higher-Higher-Order function that adds three numbers!

```
def f(x):
    """Returns a higher-order function that would add three numbers!
    >>> f(2)(3)(4)
    9
    >>> a = f(2)
    >>> a(3)(5)
    10
    >>> a = f(1)
    >>> b = a(5)
    >>> b(6)
    12
    """
```

Lambda Funcitons

Lambda expressions are one-line functions that specify two things: the parameters and the return value.

```
lambda <parameters>: <return value>
```

While both lambda and def statements are related to functions, there are some differences.

	lambda	**def**
Type	lambda is an expression	def is a statement
Description	Evaluating a lambda expression does not create or modify any variables. Lambda expressions just create new function objects without changing the current environment.	Executing a def statement will create a new function object and bind it to a variable in the current environment.
Example	lambda x: x * x	def square(x): return x * x

A lambda expression by itself is not very interesting. As with any values such as numbers, Booleans, and strings, we usually:

- assign lambdas to variables (foo = lambda x: x)
- pass them in to other functions (bar(lambda x: x))

Try understanding the following code, write what you think.

```
>>> (lambda x: x * x)(2)
4
>>> addone = lambda y: y + 1
>>> addone(2)
3
>>> add_two_numbers = lambda x, y: x + y
>>> add_two_numbers(2, 3)
5
>>> weird_add = lambda a: lambda b: a + b
>>> weird_add(4)(5)
9
```

Question: Write the lambda versions for the following code!

```
def f(x, y, z):
    return (x+y)//z



def f(a):
    def g(b):
        def h(c):
            return a + b + c
        return h
    return g
```

3 Citation and Sources

Example questions and explanations are brought from the following sources:

1. Hilfinger Paul, DeNero John, "CS 61A: Structure and Interpretation of Computer Programs." CS 61A Fall 2017, cs61a.org.

2. Point, Tutorials. "Python Numbers." www.tutorialspoint.com, Tutorials Point, 5 Oct. 2017, www.tutorialspoint.com/python/python_numbers.htm.