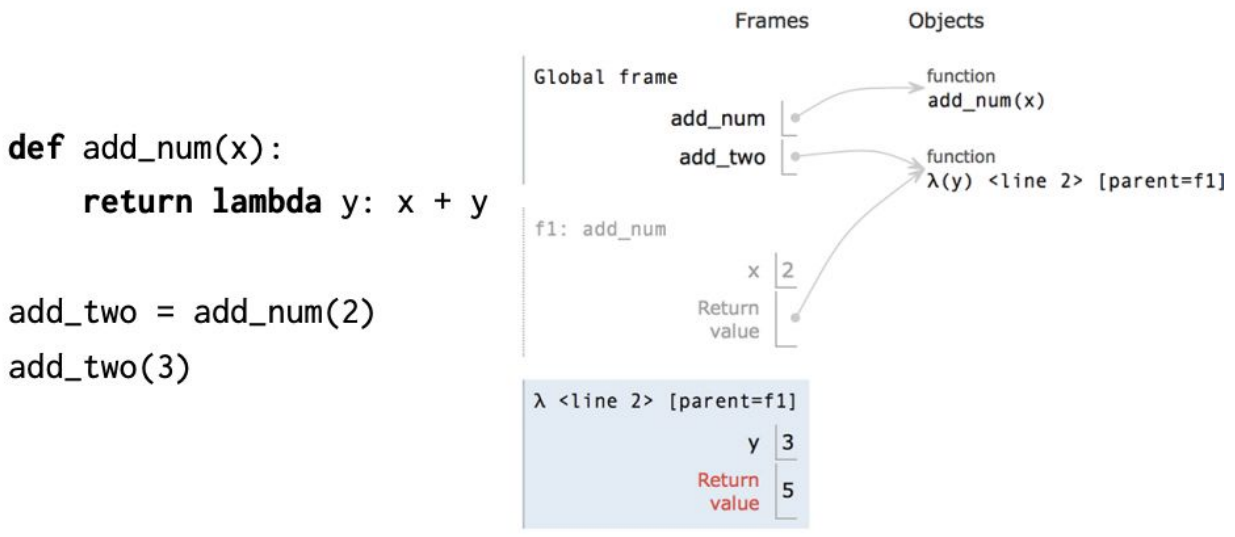
HIGHER ORDER FUNCTIONS

CS 7 NOTE 2

Compiled by Eric Khumalo

Higher Order Functions in Environment Diagrams

An environment diagram keeps track of all the variables that have been defined and the values they are bound to. However, values are not necessarily only integers and strings. Environment diagrams can model more complex programs that utilize higher order functions.



```python
def add_num(x):
    return lambda y: x + y

add_two = add_num(2)
add_two(3)
```

Lambdas are represented similarly to functions in environment diagrams, but since they lack intrinsic names, the lambda symbol **(λ)** is used instead. The parent of any function (including lambdas) is always the frame in which the function is defined. It is useful to include the parent in environment diagrams in order to find variables that are not defined in the current frame. In the previous example, when we call **`add_two`** (which is really the lambda function), we need to know what `x` is in order to compute `x + y`. Since `x` is not in the frame `f2,` we look at the frame's parent, which is `f1`. There, we find `x` is bound to `2`.

As illustrated above, higher order functions that return a function have their return value represented with a pointer to the function object.

Lambda Expressions

A lambda expression evaluates to a function, called a lambda function.

Acknowledgements: These notes are an adaptation from CS61A material at UC Berkeley.
Credits to John DeNero and the CS61A staff.

In the code above, `lambda y: x + y` is a lambda expression, and can be read as a function that takes in one parameter `y` and returns `x + y`

A lambda expression by itself evaluates to a function but does not bind it to a name. Also note that the return expression of this function is not evaluated until the lambda is called. This is similar to how defining a new function using a def statement does not execute the functions body until it is later called.

```
>>> what = lambda x : x + 5
>>> what
<function <lambda> at 0xf3f490>
```

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

Challenge Yourself!

Environment Diagrams

Now use all the knowledge from the information above and the previous notes to draw the environment diagram that results from executing the code below.

```
def curry2(h):
    def f(x):
        def g(y):
            return h(x, y)
        return g
    return f
make_adder = curry2(lambda x, y: x + y)
add_three = make_adder(3)
add_four = make_adder(4)
five = add_three(2)
```

Try challenging yourself by writing `curry2` as a lambda function

Writing Higher Order Functions

Acknowledgements: These notes are an adaptation from CS61A material at UC Berkeley.
Credits to John DeNero and the CS61A staff.

Write a function that takes in a function `cond` and a number `n` and prints numbers from `1` to `n` where calling `cond` on that number returns `True`.

```
def keep_ints(cond, n):
      """Print out all integers 1..i..n where cond(i) is true

      >>> def is_even(x):
      ...   # Even numbers have remainder 0 when divided by 2.
      …     return x % 2 == 0
      >>> keep_ints(is_even, 5)
      2
      4
      """
      ***YOUR CODE HERE***
```

Acknowledgements: These notes are an adaptation from CS61A material at UC Berkeley.
Credits to John DeNero and the CS61A staff.