Cs61a

Week 1 Lecture2 2022/11/30:

**Types of expression**

Primative expressions: number, name, string

Call expression: mul()

Mul(add(2,3), add(3,4)) :

>>>From operator import add, mull this is an import statement

Expression tree

add is operator, 2 is operand, operators and operand are also expressions

add(2,3) is operand subexpression, 5 is value of subexpression

**Assignment**

Bind name to value

Bind multiple name to multiple value:

>>>radius = 20 this is an assignment statement

>>>area, radio = radius\*30, 3+4

>>> area

600

If you change radius now, area still the same? How to change radius and keep arear updated? Function

>>>Rafius =20

>>>Def area():

Return radius\*30

>>>area()

600

>>>radius =2

>>>area()

60

**Enviroment Diagrams:** <https://pythontutor.com/cp/composingprograms.html#mode=edit>

Visualize the interpreter’s process

Global frame: pi = 3.1425 where pi is name and 3.1415 is value, they are binded

**Function:**

Bind names to expressions

Function signiture: indicate how many arguments a function takes

Function body: defines the computational process expressed by a function

**Enviroment:**

Every expression is evaluated in the context of an enviroment(either the global frame or the local frame)

An enviroment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current enviroment in which that name is found:

Eg: look up some name in the body of the square function:

Look for that name in the local frame

If not find, look for it in the global frame

>>>From operator import mul

>>>Def square(square):

Return mul(square, square)

>>>Square(-2)

Frames：

Global frame: mul, square

Local frame: square -2

This function works because we found square = -2 in the local frame at first, so we do not need to find it in the global frame

Week 2 2022/11/30

Control

A function that does not explicitly return a value will return None:

>>>Def apple(n):

n\*n

>>>x = apple(4)

>>>y = 5

>>>x+y 5 is actually bind to a None value which will give error

**Nested expressions with Print:**

Print is a non pure function which will always print what in parenthesis and return none

>>> print(print(2), print(3))

2

3

None None

**User-defined function cycle:**Notability

Devision:

>>>2013 / 10

201.3

>>>2013 // 10 (5 //3 = 1)

201

>>> 2013 % 10

3

>>> from operator import truediv, floordiv, mod

>>> truediv(2013, 10)

201.3

>>>floordiv(2013, 10)

201

>>>mod(2013, 10)

3

py -m doctest -v ex.py will show the doctext, how the function is working through

**Statement:**

A statement is excuted by the interpreter to perform an action

Higher-Order Functions

A function that takes a function as an argument value or returns a function as a return value

Prime Fractorization: each positive integar n has a set of prim factos. Primes produce n

Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34

Designing function:

A function’s **domain** is the set of all inputs it might possibly take as argument, a function’s **range** is the set of output values it might possibly return, a function’s **behavior** is the relationship it creats between input and output

Week 2 2022/12/1

Enviroments

**Enviroments for high-order functions**

>>>def apply\_twice(f,x):

Return f(f(x))

>>>def square(x):

Return x\*x

>>>result = apply\_twice(square, 2)

16

Global frame:

apply\_twice| func apply\_twice(f,x) [parent = Global]

Square| func square(x) [parent = Global]

Result|16

We got two names of function that bound to functions, but does not excuted the body

f1: apply\_twice [parent = Global]

f|square

x|2

return|16

introducing a new frame, the formal parameters f and x are bound to the arguments square function and number 2,

At which point the body ff(x)is excuted. In order to evaluate ff(x), we need to both evaluate f() operator, and operand expression f(x)

f2: square [parent = Global]

x|2

return|4

f(x) requires a call of f(), which is currently bound to square, so we call the square function on the number 2, which is what x is bound to. We end up a return value of 4. So the inner f(x) equal to 4.

f3: square [parent = Global]

x|4

return|16

Now we call f() on 4 and return value of 16, which is eventually bound to the name result

Applying a user-defined function:

* Creat a new frame:f1, f2, f3
* Bind formal parameters f, x to arguments
* Excute the body: return f(f(x)

………………………………………………………………………………

**Enviroments for nested definitions**

>>>def make\_adder(n):

def adder(k):

Return k +n

Return adder

>>>add\_three = make\_adder(3)

>>>result = add\_three(4)

Global frame:

make\_adder: func make\_adder(n) [parent = G]

result|7

f1: make\_adder [parent = G]

n|3

adder|func adder(k) [parent = f1]

return| func adder(k) [parent = f1]

we call *make\_adder* function which intoducing a new fram f1 with formal parameter n bound to argument 3

next we define adder() function which is th first step in the body of the make\_adder() function, when we define the adder function, we creat a new function, then we bound the name *adder* to the new function value *func adder(k)* in the current frame. So the make\_adde frame and its enviroment have access to this adder function. Return will refer the fnction back to the global frame. Bound to the name add\_three

f2: adder [parent = f1]

k|4

return|7

now call sdd\_three on 4 so we inreoduce a new frame with k bind to 4. We evaluate the expression k plus n in an enviroment that starts with f2 followed by f1 followed by global frame. We find k=4 and find n=3, then return 7

Every user-defined function has a parent frame(often global)

The parent of a function is the frame in which it was defined

Every local frame has a parent frame

The parent of a frame is the parent of the function called

………………………………………………………………….

**Enviroment disgram for function composition**

>>>def square(x):

Return x \*x

>>>def make\_adder(n):

def adder(k):

Return k+n

Return adder

>>>def compose1(f, g):

def h(x):

Return f(g(x))

Return h

>>>compose1(square, make\_adder(2))(3)

global frame:

square|func square(x) [parent = G]

make\_adder|func make\_adder(n) [parent = G]

compose1|func compose1(f, g) [parent = G]

f1: make\_adder [parent = G]

n|2

adder|func adder(k) [parent = f1]

return|func adder(k) [parent = f1]

call compose1 will call make\_adder(), which will introduce a new frame f1 with 2 bind to n, then define a new function adder() and return the function

f2: compose1 func compose1(f, g) [parent = G]

f| func square(x) [parent = G]

g| func adder(k) [parent = f1]

h| func h(x) [parent =f2]

return| func h(x) [parent =f2]

we return that to compose1, the compose1 has parameter f bind to argument square, g binds to adder(), then define h and retuen it

f3:h [parent = f2]

x|3

f4: adder(k) [parent = f1]

k|3

retuen|5

f5: square(x) [parent = G]

x|5

retuen|25

we call h(x), which is going to call f of g of x, first we will compute g of x which is calling adder function where k is bound to 3, and 2 is 2, so that give a return of 5. Which will be passed to square, which return 25, so the answer to h is 25

………………………………………………………………………………………………………………………………………………………………

**How to draw an enviroment diagram:**

When a function is defined:

Create a function value: func <name> (<formal parameters>) [parent=<parent>]

Its parent is the current frame

We bind <name> to the function value in the current frame

When a function is called:

1. add a local frame, titled with the <name> of the function being called.
2. Copy the parent of the function to the local frameL [pparent=<label>]
3. Bind the <formal parameters> to the arguments in the local frame
4. Execute the body of the function in the enviroment that starts with the local frame

**Function Currying**

>>>def make\_adder(n):

def adder(k):

Return k+n

Return adder

>>>def make\_adder(n):

Return lambda k: n +k

>>>def curry2(f):

def g(x):

def h(y):

return f(x,y)

return h

return g

>>>add(2,3)

5

>>>m = curry2(add)

>>>add\_three = m(3)

>>>add\_three(2)

5

>>>add\_three(2010)

2013

Curry2 = lambda: f: lambda x: lanbda y: f(x,y)

Curry2 is a nested lambda expression, it is a function that takes f return a function that takes x return a function that takes y that takes f(x,y)

………………………………………………………………………………………………………………

Lab01

Q: How do you prevent the ok autograder from interpreting print statements as output?

Choose the number of the correct choice:

Print with 'DEBUG:' at the front of the outputted line

Q: What is the best way to open an interactive terminal to investigate a failing test for question sum\_digits in assignment lab01?

Choose the number of the correct choice:

2) python3 ok -q sum\_digits -i

Enter will take you from command mode back into edit mode for the given cell.

Esc will take you into command mode where you can navigate around your notebook with arrow keys.

While in command mode:

A to insert a new cell above the current cell, B to insert a new cell below.

M to change the current cell to Markdown, Y to change it back to code

D + D (press the key twice) to delete the current cell

C copy

V paste

Y change the cell type to code

M change the cell type to markdown

Ctrl + Shift + - will split the current cell into two from where your cursor is.

Shift + M to merge multiple cells.

Shif + up, select more than one raw

Esc + a: insert raw above

Esc + b: insert raw below