CS61A NOTE1 Control, Environment

Control structures

Conditional statements 条件

if-elif-else

truthy value (`True` , a non-zero integer 非零值.) or a falsy value (`False` , `0` , `None` , `""` , `[]` , ...)

1 if, n elif,1/0 else

A series of `if` statements has the same effect as using both `if` and `elif` cases if each `if` clause ends in a `return` statement.每次 if 重启新的判定

Boolean Operators 布尔

not

and

meet first false—short-circuits and return it, all true—return last value

or

meet first true—short-circuits and return it, all false—return last value

While loops 循环

```
#discussion1 Q4
```

```
def is_prime(n): #是质数吗？
    if n == 1: #除开n为1的特例
        return False
    k = 2
    while k < n:
        if n % k == 0:
            return False  #return可以跳出while循环
        k += 1
    return True  #前面一直没跳出，最终return True
```

be careful about the order in which they have their `if-elif` statements 注意顺序

Evironment Diagrams 环境图

An environment diagram is a model we use to keep track of all the variables that have been defined and the values they are bound to. 环境图是我们用来跟踪所有已定义的变量和它绑定值的模型。One key idea in environment diagrams is the frame. A frame helps us keep track of what variables have been defined in the current execution environment, and what values they hold. The frame we start off with when executing a program from scratch is what we call the Global frame. 环境图中的一个关键概念是 frame。frame 可以帮助我们跟踪当前执行环境中已定义的变量以及它持有的值。当我们从头开始执行一个程序时，我们开始的 frame 就是我们全局 Global frame。

Assignment Statements 赋值语句=

Evaluate the expression on the right side of the `=` sign. Write the variable name and the expression's value in the current frame.计算等号右侧的表达式，在当前框架中写下变量名和表达式的值。若 frames 没有左式，新增 frame 并与 object 右式相连。若已有左式，可以划掉之间连线并连接最新 objects 右式。

def starement 定义 def

"def 创建一个函数对象并将其绑定到一个名称。为了绘制 def 语句的图示，记录函数名称并将函数对象绑定到该名称上。写下函数的父级框架也很重要(函数定义的地方)。Assignments for `def` statements use pointers to functions, which can have different behavior than primitive assignments.def 语句的赋值使用函数指针，其行为可能与原始赋值不同。Draw the function object to the right-hand-side of the frames, denoting the intrinsic name of the function, its parameters, and the parent frame.将函数 object 绘制在右侧，表示函数的内部名称、参数和 parent frame。 Write the function name in the current frame and draw an arrow from the name to the function object.在当前 frame 中写下函数名称，并从此名称到函数 object 绘制箭头。

Call Expressions 调用

Call expressions, apply functions to arguments. When executing call expressions, we create a new frame in our diagram to keep track of local variables 调用将参数传入函数。执行调用表达式时创建一个新 frame

1. Evaluate the operator 运算符, which should evaluate to a function.
2. Evaluate the operands from left to right.从左到右算要传入的数 operand
3. Draw a new frame, labelling it with the following:
    - A unique index (`f1` , `f2` , `f3` , ...).索引
    - The **intrinsic name** of the function, which is the name of the function object itself. 写在左侧的函数内在名称是函数右侧 object 名称
    - The parent frame ([`parent=Global`]).
4. Bind the formal parameters to the argument values obtained in step 2 (e.g. bind `x` to 3).将形参绑定到步骤 2 中获得的参数值
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.在此新 frame 中评估函数直到获得返回值，frame 中写下返回值。

If a function does not have a return value, it implicitly returns `None` . In that case, the “Return value” box should contain `None` .没有返回值 None

Note: Since we do not know how built-in functions like `min(...)` or imported functions like `add(...)` are implemented, we do not draw a new frame when we call them, since we would not be able to fill it out accurately.

Lambda Expressions

A lambda expression evaluates to a function, called a lambda function. lambda 表达式被求值为一个函数 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 function's body until it is later called.一个 lambda 表达式本身被求值后会得到一个函数，但并没有将其绑定到一个名字。另外，该函数的返回表达式直到 lambda 函数被调用 call 时才会被求值。类似 def ，在函数被调用之前并不执行函数体。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. 与 def 语句不同，lambda 表达式可以用作运算符或操作数传递给调用表达式。它们只是一行表达式，是函数。

Higher Order Function

A **higher order function** (HOF) is a function that manipulates other functions by taking in functions as arguments, returning a function, or both. 将函数作为参数、返回函数或两者兼而有之来操纵其他函数的函数

Lambdas are represented similarly to functions in environment diagrams, but since they lack instrinsic names, the lambda symbol (λ) is used instead. λ 没有 instrinsic name 因此 object 直接记为 λ. The parent of any function 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. As illustrated above, higher order functions that return a function have their return value represented with a pointer to the function object. HOF 的父级可能是 fi,去父级找未知的参数

Q5 make keeper

Write a function that takes in a number `n` （输入 n） and returns a function that can take in a single parameter `cond` （接受 1 参数的函数） When we pass in some condition function `cond` into this returned(返回函数传入 cond) function, it will print out numbers（打印 1–n 中 True 的） from 1 to `n` where calling `cond` on that number returns `True`.

```
def make_keeper(n):
    def helper(cond):
        i=1
        while i <= n: #输入了n
            if cond(i):
                print ('i') #print的用法
            i+=1
    return helper #不用参数
```

Curring

One important application of HOFs is converting a function that takes multiple arguments into a chain of functions that each take a single argument. This is known as **currying**. HOF 将一个需要多个参数的函数转换为一连串各需要一个参数的函数，即 currying。

Q6 Curring

Write a function `curry` that will curry any two argument function.

```
def curry(func): #传入curry的是func，传入func(n1,n2)的是n1,n2
    def curried_func(n1): #两个参数，def两个函数
        def func_n2(n2):
           return func(n1,n2)
        return func_n2 #return后不用加参数
    return curried_func
```

Q7 Make my own lambdas

use lambdas instead of nested def statements.

```
def f1():
    """
    >>> f1()
    3
    """
    return 3

def f2():
    """
    >>> f2()() #后1个()因此return的是1个参数函数，任意值都3
    3
    """
    return lambda:3

def f3():
    """
    >>> f3()(3) #后1个()因此return的是1个参数函数，x映射x
    3
    """
    return lambda x:x

def f4():
    """
    >>> f4()()(3)() #后3个()因此return的是3个参数函数，连写lambda
    3
    """
    return lambda: lambda x: lambda: x
```

纯文本

Q8 Lambdas and Currying

Write a function that will curry any two argument function like with curry, but this time using lambdas. 用 lambda 而非定义两个 def 两个参数的函数

```
def lambda_curry2(func):
    return lambda n1: lambda n2: func(n1,n2)
```

纯文本

Q9: Match Maker

Implement `match_k`, which takes in an integer `k` and returns a function that takes in a variable `x` and returns `True` if all the digits in `x` that are `k` apart are the same(x 中相隔 k 数位数字相同).

纯文本

```
def match_k_alt(k): #没输入x,因此def新的函数
    def helper(x):
        while x//(10**k):
            if (x%10) != (x//(10**k))%10:
                return False
            x=x//10
        return True
    return helper
```