Discussion 2

Environment Diagrams and Higher-Order Functions

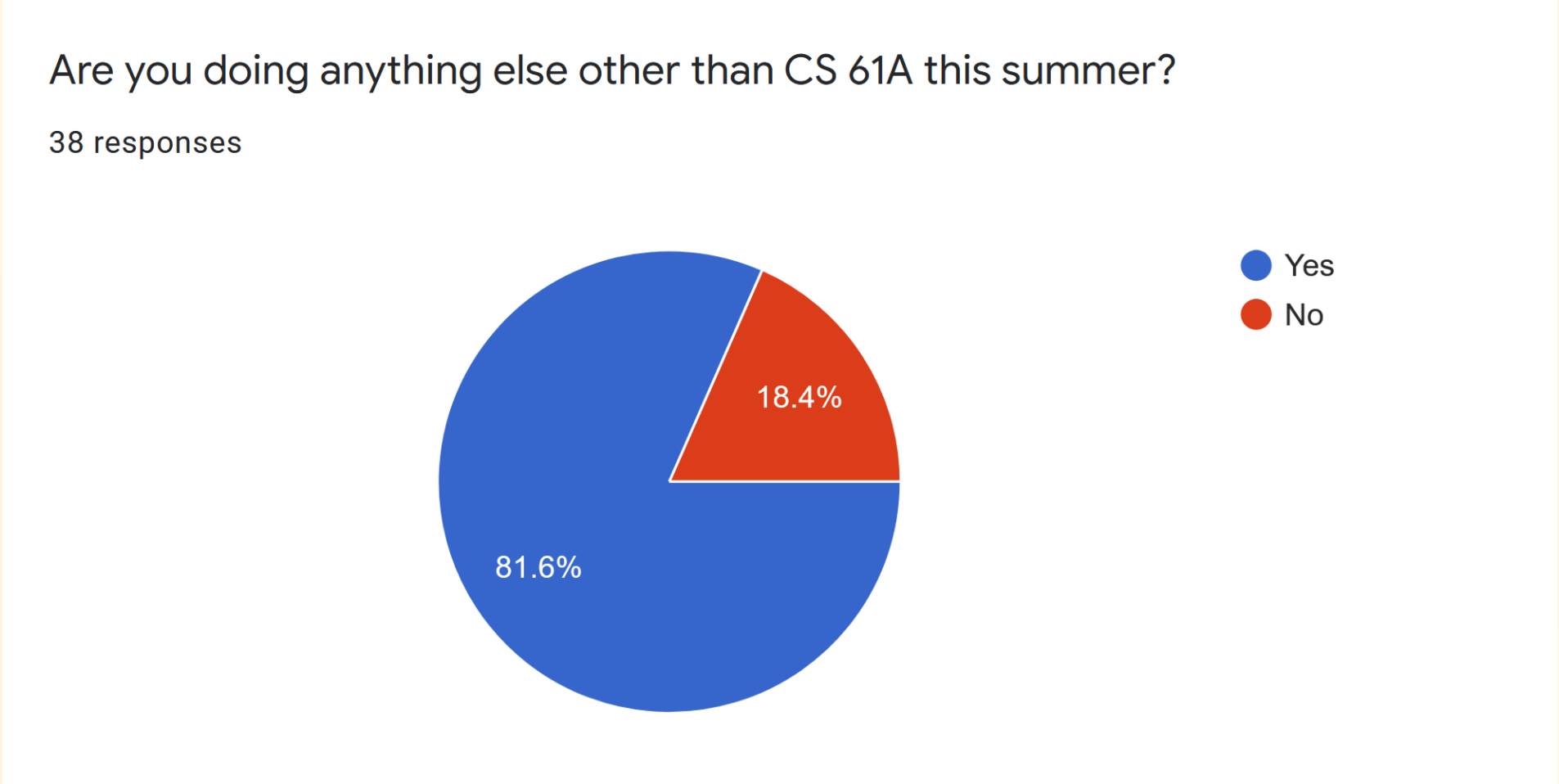
Antonio Kam

anto [at] berkeley [dot] edu

Announcements

- Hog released!
  - Due July 6th (Next Wednesday)
  - Checkpoint
    - Submit with Phase 1 complete by Friday (July 1st) for 1 pt.
  - Extra Credit, 1 pt, (cannot use extension on this)
    - Highly recommend doing it
    - Submit the entire project on July 5th
  - Solo project! (Other projects will allow collaboration)
- HW 1 released!
  - Due Thursday (6/30)
- Tutoring Sections have opened!

Questions and Comments from last discussion



Questions and Comments from last discussion

- Assignments are always due at `11:59 PM` on the specified due date
- Lab 0 Attendance - if you didn't get attendance then, you're fine - I just put attendance up on the board as practice
  - One thing worth noting is that Lab 0 does **not** count for lab attendance!
  - This is similar for discussion 0, but doesn't matter as much!
- Ori is a good game
  - The music is partially what got me into orchestral arranging 👀
- CS 61A runs at double the speed over the summer - the workload might seem quite intense, and that's because it is! You're doing 2 weeks worth of material in 1 week.

Temperature Check 🌡️

- Environment Diagrams
- `lambda` functions
- Higher-order Functions

All slides can be found on

`teaching.rouxl.es`

Environment Diagrams 

Environment Diagrams

- Environment diagrams are a great way to learn how coding languages work under the hood
- Keeps track of all the variables that have been defined, and the values that they hold
  - Done with the use of *frames*
- Expressions evaluate to values:
  - `1 + 1` → `2`
- Statements do not evaluate to values:
  - `def` statements, assignments, etc.
- Statements change our environment

Frames

- The `Global Frame` exists by default
- Frames list bindings between variables and their values
- Frames also tell us how to look up values

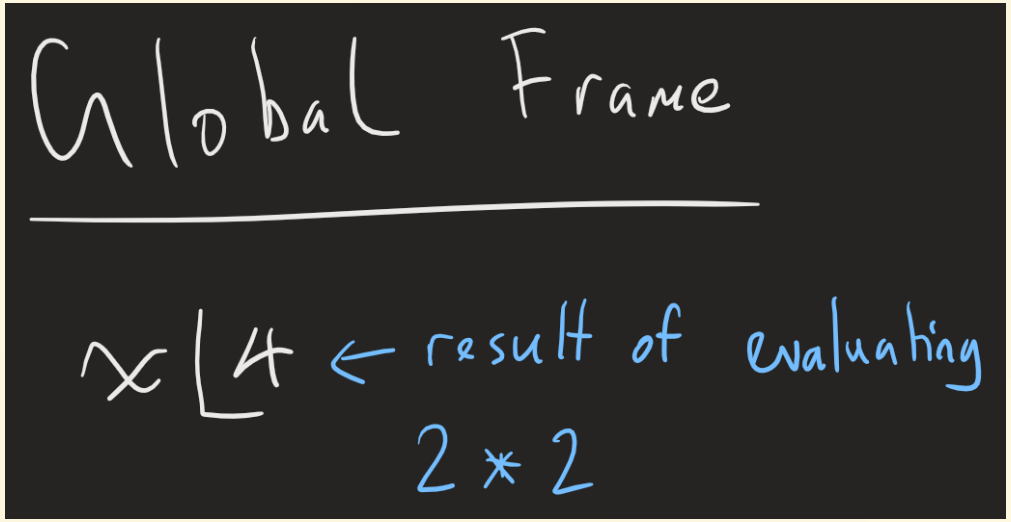
Assignment

- Assignment statements bind a value to a name
  - The right side is evaluated before being bounded to the name on the left
  - `=` is not the same in Python and mathematics
- These are then put in the *correct frame* in the environment diagram

```
x = 2 * 2 # 2 * 2 is evaluated before bound to the name x
```

Assignment

```
x = 2 * 2 # 2 * 2 is evaluated before bound to the name x
```



`def` statements

- Creates function (objects), and binds them to a variable name
- The function is **not** executed until called!
- Name of the variable is the name of the function
- Parent of the function is the frame where the function is *defined*
- Keep track of:
  - Name
  - Parameters
  - Parent

Example

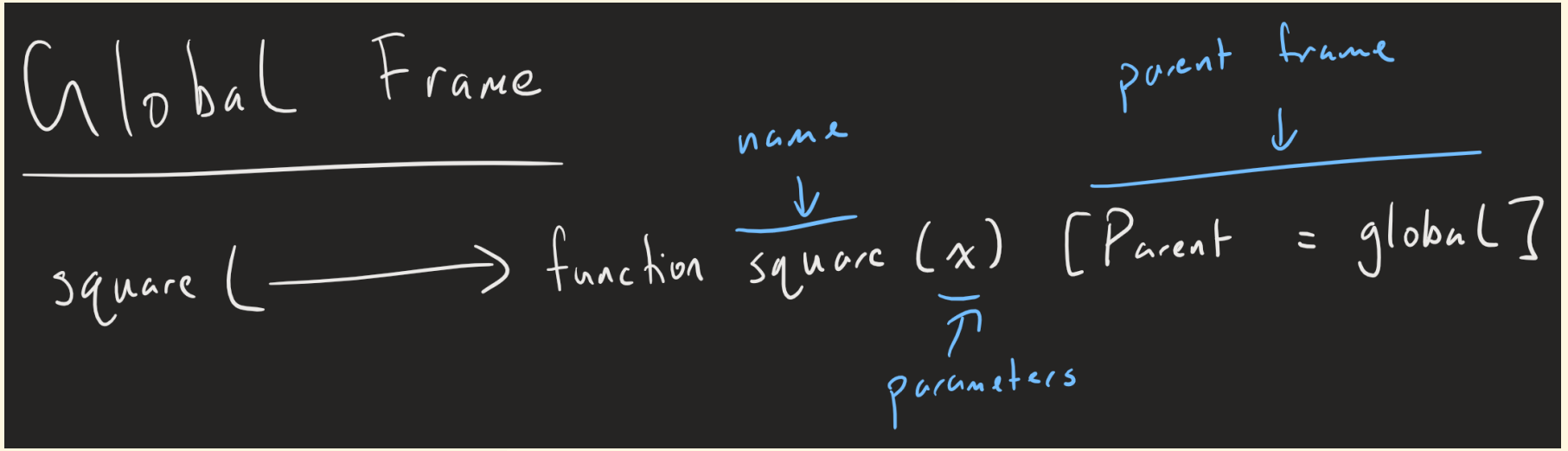
```
def square(x):
  return x * x
```

- Keep track of the name, parameters, and parent!
- Uses *pointers* (unlike for primitive values)

Example

```
def square(x):
  return x * x
```

- Keep track of the name, parameters, and parent!
- Uses *pointers* (unlike for primitive values)



Call Expressions

Example 1

```
add(5, 9) # 14
```

Example 2

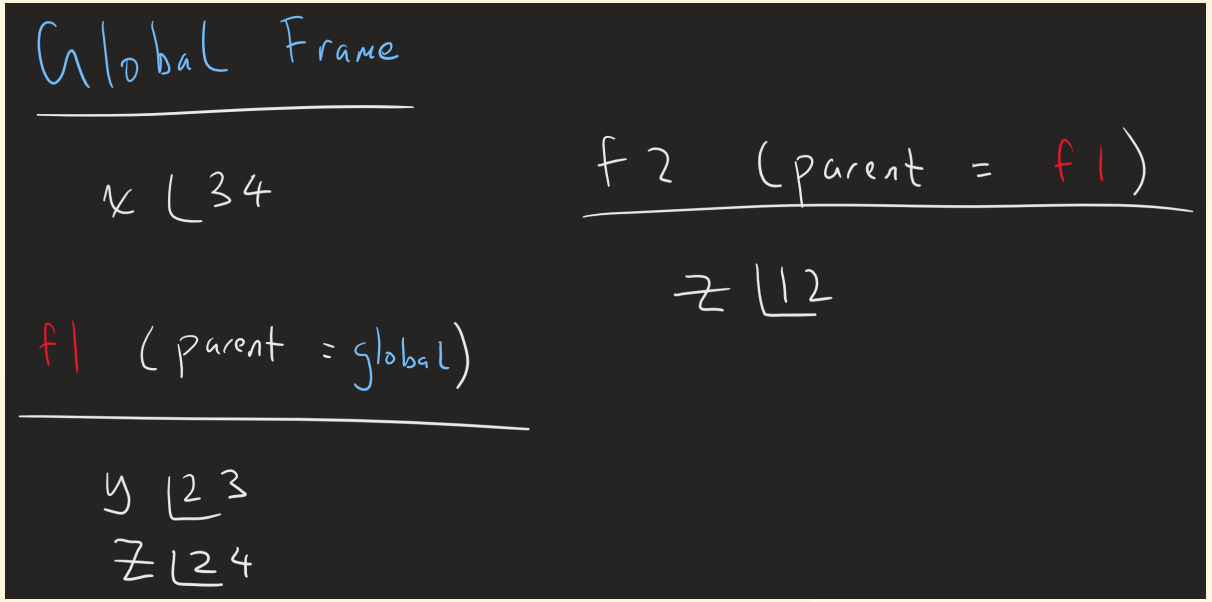
```
x = 3
add(2, add(x, 4)) # 9
```

Variable Lookup ☝️

- Look in your current frame to find your variable
- If it doesn't exist, repeat the same process in the parent frame (including the lookup if you don't find anything)
- If you reach the global frame and still can't find anything, the program errors
  - This is because the variable doesn't exist 😭

Variable Lookup

Example



(Assume that we're looking for variables inside `f2`)

Variable Lookup

Example

Variable	Value
x	34
y	23
z	12

- If we start off in `f2`, we already see `z` in `f2`, so there is no need to look at the frame above.
- However, for the case of `y`, we do need to look up to its parent frame, and for `x`, we need to lookup 2 levels

New Frames

- Label your frame with a unique index (convention is `f1`, `f2`, etc.)
- Write down the name of the function object
  - Not necessarily the name of the variable!
- Write down the parent that the function you're calling has
- Separately, all frames (other than the global frame) have a return value
  - This can be `None` if nothing is specified

Example

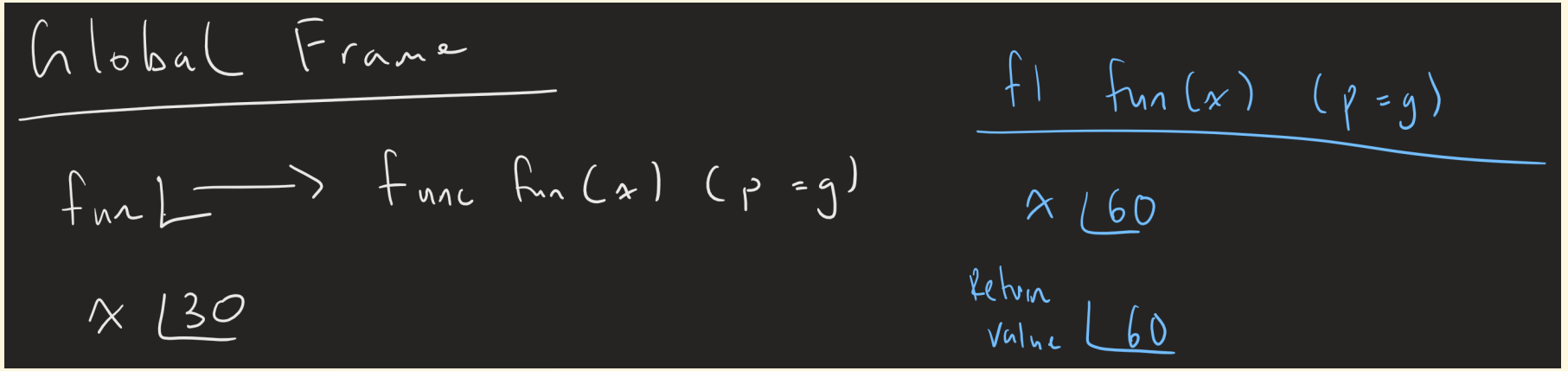
```
def fun(x):
  x = x * 2
  return x

x = 30
fun(x)
```

Example

```
def fun(x):
  x = x * 2
  return x

x = 30
fun(x)
```



Question 1 (5 minutes)

Draw the environment diagram for the following

```
def double(x):
    return x * 2

hmmm = double
wow = double(3)
hmmm(wow)
```

Attendance

`links.rouxl.es/disc`

Question 2 (Walkthrough)

```
def f(x):
    return x

def g(x, y):
    if x(y):
        return not y
    return y

x = 3
x = g(f, x)
f = g(f, 0)
```

`lambda` Functions and Higher-Order Functions

- A `lambda` expression evaluates to a `lambda` function
  - Can be used as the operator for a function!
- These functions work the same way as a normal function
  - Can be written in 1 line - faster way to make functions
  - Similar to `def` in usage, but different syntax
- `lambda`s are especially useful when you want to use a function once and then never use it again (will see examples of this)

`lambda` Syntax

- `lambda <args>: <body>`
- What goes in `<body>` must be a single expression

`lambda` Example

```
def func(x, y):
  return x + y

func = lambda x, y: x + y
# Notice how I have to do the binding to a variable myself
```

```
def i(j, k, l):
  return j * k * l

i = lambda j, k, l: j * k * l
```

`lambda` Example 2

`lambda` functions can also be used as the operator for a function!

```
(lambda x, y: x + y)(2, 3) # 5

# Equivalent to

def add(x, y):
  return x + y

add(2, 3) # 5
```

Higher Order Functions (HOF)

- HOFs are functions that can do the following things (can be both):
  1. Take in other functions as inputs
  2. Return a function as an output
- You can treat a function as just an object or a value (there's nothing special about them)
- `function` and `function()` mean different things!
  - `function` refers to the object itself (in the environment diagram, it refers to what the arrow is pointing to)
  - `function()` actually calls and executes the body of the function

HOF Example 1 (Functions as input)

```
def double(x):
  return x * 2

def square(x):
  return x ** 2

def double_adder(f, x):
  return f(x) + f(x)

double_adder(double, 3) # 12
double_adder(square, 3) # 18
# Passed in two different functions
```

HOF Example 2 (Functions as output)

```
def f(x):
  def g(y):
    return x + y
  return g

a = f(2)
a(3) # 5

# Same thing as calling f(2)(3)
```

HOF Example 2

```
def f(x):
  def g(y):
    def h(z):
      return x + y + z
    return h
  return g

lambda x: lambda y: lambda z: x + y + z
```

The two above are equivalent statements!

(Notice how the lambda one takes up far less space!)

Question 3 (5 minutes)

Draw the environment diagram for the following code and predict what Python will output.

```
a = lambda x: x * 2 + 1
def b(b, x):
    return b(x + a(x))
x = 3
x = b(a, x)
```

Question 4 (5 minutes)

Draw the environment diagram for the following code and predict what Python will output.

```
n = 9
def make_adder(n):
    return lambda k: k + n
add_ten = make_adder(n+1)
result = add_ten(n)
```

- In the Global frame, the name `add_ten` points to a function object. What is the *intrinsic* name of that function object, and what frame is its parent?
- What name is frame f2 labeled with (`add_ten` or λ)? Which frame is the parent of f2?
- What value is the variable result bound to in the Global frame?

Question 5 (10 minutes)

Write a function that takes in a number `n` and returns a function that can take in a single parameter `cond`. When we pass in some condition function `cond` into this returned function, it will print out numbers from `1` to `n` where calling `cond` on that number returns `True`.

```
def make_keeper(n):
    """Returns a function which takes one parameter cond and prints
    out all integers 1..i..n where calling cond(i) returns True.

    >>> def is_even(x):
    ...     # Even numbers have remainder 0 when divided by 2.
    ...     return x % 2 == 0
    >>> make_keeper(5)(is_even)
    2
    4
    """
```

Question 5

```
def make_keeper(n):
    """Returns a function which takes one parameter cond and prints
    out all integers 1..i..n where calling cond(i) returns True.

    >>> def is_even(x):
    ...     # Even numbers have remainder 0 when divided by 2.
    ...     return x % 2 == 0
    >>> make_keeper(5)(is_even)
    2
    4
    """
    def keeper(cond):
      i = 1
      while (i <= n):
        if cond(i):
          print(i)
        i += 1
    return keeper # remember this line!
```

Currying

Currying is one application of the HOFs from earlier.

```
lambda x: lambda y: x + y
```

Instead of just any expression on the inside (for example `x + y`), we use a function!

```
def pow(x, y):
  x ** y

def curried_pow(x):
  def f(y):
    return pow(x, y)
  return f

curried_pow(3)(2)
# is the same as
pow(3, 2)
# You will need as many inner functions as you have arguments
```

Currying

- Currying is the process of turning a function that takes in *multiple* arguments to one that takes in *one* argument.
- What's the point?
  - Sometimes functions with 1 argument are far easier to deal with
  - Can create a bunch of functions that have slightly different starting values which saves on repeating code
- Kind of hard to see the benefits until you write production code

Question 7

Draw the environment diagram that results from executing the code below.

```
n = 7

def f(x):
    n = 8
    return x + 1

def g(x):
    n = 9
    def h():
        return x + 1
    return h

def f(f, x):
    return f(x + n)

f = f(g, n)
g = (lambda y: y())(f)
```

Question 8

```
def match_k(k):
    """ Return a function that checks if digits k apart match

    >>> match_k(2)(1010)
    True
    >>> match_k(2)(2010)
    False
    >>> match_k(1)(1010)
    False
    >>> match_k(1)(1)
    True
    >>> match_k(1)(2111111111111111)
    False
    >>> match_k(3)(123123)
    True
    >>> match_k(2)(123123)
    False
    """
```

Question 8

```
def match_k(k):
    """Return a function that checks if digits k apart match"""
    ____________________________
        ____________________________
        while _______________________________:
            if ____________________________:
                return ____________________________
            ____________________________
        ____________________________
    ____________________________
```

Question 8

```
def match_k(k):
    """Return a function that checks if digits k apart match"""
    def matcher(n):
        while n // (10 ** k) > 0:
            if n % 10 != (n // (10 ** k)) % 10:
                return False
            n //= 10
        return True
    return matcher
```

Mental Health Resources

- CAPS:
  - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
  - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!