Tutorial: Week 2 (9/2/20)

Question 1:

Tutorial: Write a function that returns `True` if a positive integer `n` is a prime number and `False` otherwise.

A prime number n is a number that is not divisible by any numbers other than 1 and n itself. For example, 13 is prime, since it is only divisible by 1 and 13, but 14 is not, since it is divisible by 1, 2, 7, and 14.

Hint: use the `%` operator: `x % y` returns the remainder of `x` when divided by `y`.

```
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    """
```

- What should it return?
  string? number? boolean? nothing?
  boolean — True: prime; False: not prime
- What is a prime number?
  itself & one, one is NOT prime
- What might we use? what control?
  while! 2
- Possible edge cases? if n == 1 return False

n % j == 0

while (k < n)
5 if (5 % 2 == 0)
↳ return False

CHECKING ANSWER: step through code line by line writing variables / using print statements

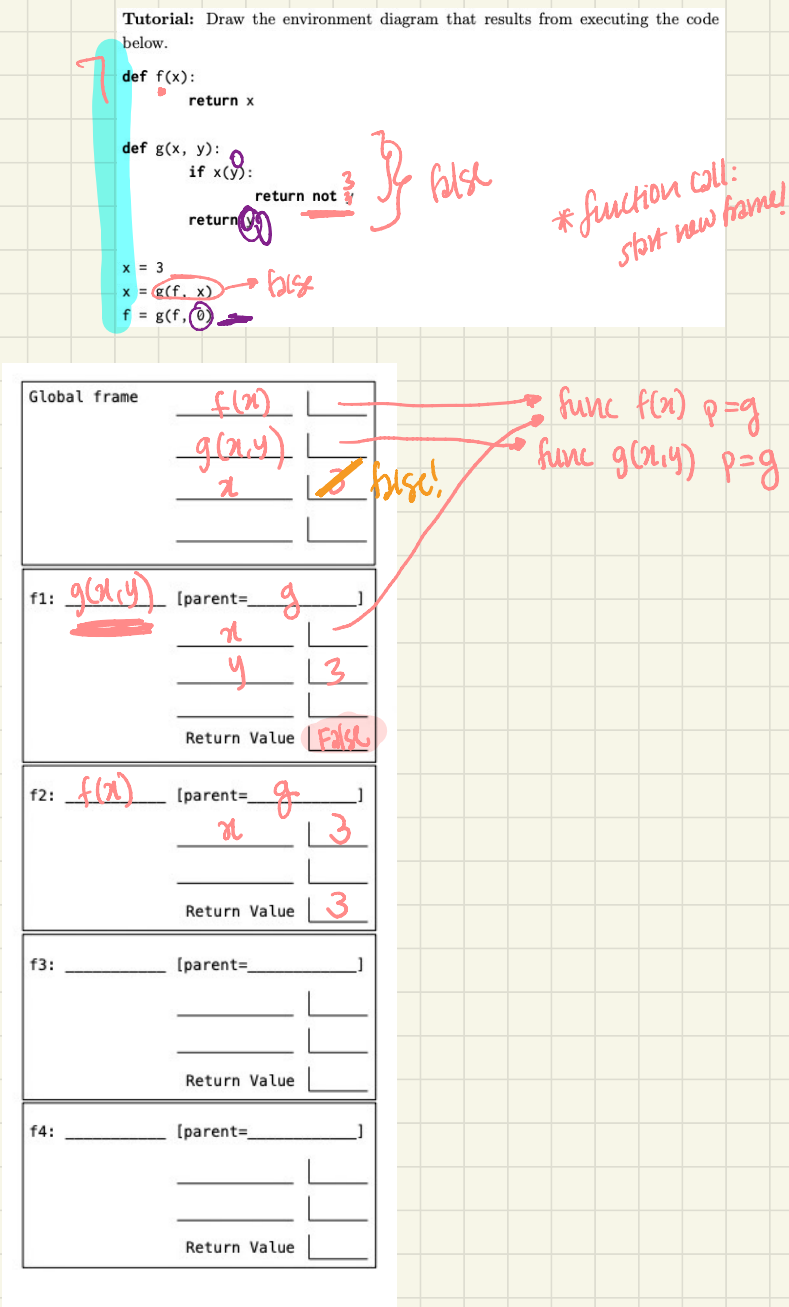
Question 2:

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

```
def f(x):
    return x

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

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



Section #2: Function, Control

Control

Call Expression: `add ( 2, 3 )` — `add` is the operator, `2, 3` are the operands

1. evaluate operator and operand
2. apply function to values

Define Function:

```
def <name> ( <formal parameter> )
```

- `<name>`: function name
- `<formal parameter>`: function parameters

```
return <return expressions>
```

- `<return expressions>`: what function returns

Environment Diagrams:

- first start with Global Frame outlining just variable and function names
- only if call to function is made, do you open a new frame

```
x = 3
y = 1 + 2 + 3
def square(x):
    return x * x
square(y)
```

Global Frame

x	3
y	6
square	→ function square (p=9)

function square (p=9)

x	6	(y → 6)
rv	36	(6*6)

* replace w/ most exact value

Print Vs. Return

- **Print:** will display value but will return None
- **Return:** returns the proper value

} understand this!

```
print(4)
```

→ although prints 4, return value is None!

If:

```
if blah:          ← checks this ①
    do this
elif blah2:       ← if above isn't true checks this ②
    do that
else:             ← if all cases aren't true, does this ③
    do thisback;
```

While:

```
while (a conditional):   ← checks if this conditional is correct
    ~ does all of this   ← will keep doing everything inside until while condition is true
      multiple times ~
return blah
```

* can create infinite loops if not careful! (when conditional always true)

```
def fun(fun):
    def time(time):
        return fun(x)
    x = 4
    return time

def boo(x):
    return x**2
    x = 5

result = fun(boo)(10)
```

time