Practice 02 - Control and Environment

Instructions: Solve the questions on the paper first. You can then verify your solution using Python shell.

Part 1 - Control

Practice 1: What Would Python Display?

```
>>> def xk(c, d):
...     if c == 4:
...         return 6
...     elif d >= 4:
...         return 6 + 7 + c
...     else:
...         return 25
>>> xk(10, 10)

_______
>>> xk(10, 6)

_______
>>> xk(4, 6)

_______
>>> xk(0, 0)

_______
```

Python 3.8.5 Shell

```
>>> def xk(c, d):
        if c == 4:
                return 6
        elif d >= 4:
                return 6 + 7 + c
        else:
                return 25

>>> xk(10, 10)
23
>>> xk(10, 6)
23
>>> xk(4, 6)
6
>>> xk(0, 0)
25
>>> |
```

Ln: 21 Col: 4

```
>>> def how_big(x):
...     if x > 10:
...         print('huge')
...     elif x > 5:
...         return 'big'
...     elif x > 0:
...         print('small')
...     else:
...         print("nothin'")
>>> how_big(7)

_______
>>> how_big(12)

_______
>>> how_big(1)

_______
>>> how_big(-1)

_______
```

Python 3.8.5 Shell

```
>>> def how_big(x):
        if x > 10:
                print('huge')
        elif x > 5:
                return 'big'
        elif x > 0:
                print('small')
        else:
                print("nothin'")

>>> how_big(7)
'big'
>>> how_big(12)
huge
>>> how_big(1)
small
>>> how_big(-1)
nothin'
>>>
```

Ln: 46 Col: 4

```
>>> n = 3
>>> while n >= 0:
...     n -= 1
...     print(n)

_____
```

Python 3.8.5 Shell

```
>>> n = 3
>>> while n >= 0:
        n -= 1
        print(n)

2
1
0
-1
>>>
```

Ln: 60 Col: 4

```
>>> positive = 28
>>> while positive:
...   print("positive?")
...   positive -= 3

_______ The "positive?" will repeat forever
```

Python 3.8.5 Shell

```
>>> positive = 28
>>> while positive:
          print("positive?")
          positive -= 3

positive?
positive?
positive?
positive?
positive?
```

Ln: 74 Col: 9

Hint: Make sure your `while` loop conditions eventually evaluate to a false value, or they'll never stop! Typing `Ctrl-C` will stop infinite loops in the interpreter

Practice 2: Boolean Expressions

```
>>> True and 13

_____
>>> False or 0

_____
>>> not 10

_____
>>> not None

_____
>>> not 0

_____
>>> (1 + 1) and 1

_____
>>> 1/0 or True

_____
>>> (True or False) and False
_____
```

Python 3.8.5 Shell

```
>>>
>>> True and 13
13
>>> False or 0
0
>>> not 10
False
>>> not None
True
>>> not 0
True
>>> (1 + 1) and 1
1
>>> 1/0 or True
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    1/0 or True
ZeroDivisionError: division by zero
>>> (True or False) and False
False
>>>
```

Ln: 203 Col: 4

```
>>> True and 1 / 0 and False

_____
>>> True or 1 / 0 or False

_____
>>> True and 0

_____
>>> False or 1

_____
>>> 1 and 3 and 6 and 10 and 15

_____
>>> 0 or False or 2 or 1 / 0

_____
```

Python 3.8.5 Shell

```
>>>
>>> True and 1 / 0 and False
Traceback (most recent call last):
  File "<pyshell#66>", line 1, in <module>
    True and 1 / 0 and False
ZeroDivisionError: division by zero
>>> True or 1 / 0 or False
True
>>>
>>> True and 0
0
>>> False or 1
1
>>> 1 and 3 and 6 and 10 and 15
15
>>> 0 or False or 2 or 1 / 0
2
>>>
```

Ln: 226 Col: 4

Practice 3: What is the result of evaluating the following codes?

```
def square(x):
    print("here!")
    return x * x

def so_slow(num):
    x = num
    while x > 0:
        x = x + 1
    return x / 0

square(so_slow(5))
```

```
>>> positive = -9
>>> negative = -12
>>> while negative:
...     if positive:
...         print(negative)
...     positive += 3
...     negative += 3

_______

_______

_______
```

Answer: the while loop is a infinity loop because x = 5, x = x+1 will always greater than 0.

Python 3.8.5 Shell

```
>>> def square(x):
        print("here!")
        return x * x

>>> def so_slow(num):
        x = num
        while x > 0:
            x = x + 1
        return x / 0

>>> square(so_slow(5))
Traceback (most recent call last):
  File "<pyshell#93>", line 1, in <module>
    square(so_slow(5))
  File "<pyshell#92>", line 4, in so_slow
    x = x + 1
KeyboardInterrupt
>>> |
```

Ln: 264 Col: 4

Python 3.8.5 Shell

```
>>> positive = -9
>>> negative = -12
>>> while negative:
        if positive:
            print(negative)
        positive += 3
        negative += 3


-12
-9
-6
>>>
```

Ln: 279 Col: 4

Part 2 - Environment Diagram

Practice 5: def statements create function objects and bind them to a name. To diagram def statements, record the function name and bind the function object to the name.
It's also important to write the parent frame of the function, which is where the function is defined.

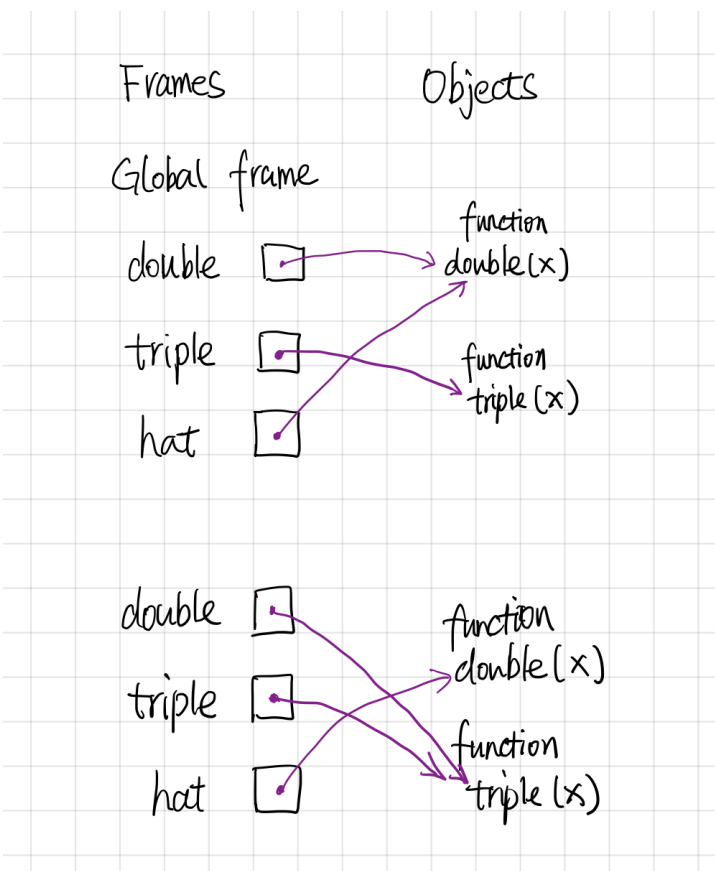
1. Draw the function object to the right-hand-side of the frames, denoting the intrinsic name of the function, its parameters, and the parent frame
   (e.g. func square(x) [parent = Global].
2. Write the function name in the current frame and draw an arrow from the name to the function object.

Use these rules and the rules for assignment statements to draw a diagram for the code below.

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

def triple(x):
    return x * 3

hat = double
double = triple
```



Practice 6: Call expressions, such as square(2), apply functions to arguments. When executing call expressions, we create a new frame in our diagram to keep track of local variables:

1. Evaluate the operator, which should evaluate to a function.
2. Evaluate the operands from left to right.
3. Draw a new frame, labelling it with the following:
    - A unique index (f1, f2, f3, ...)
    - The intrinsic name of the function, which is the name of the function object itself. For example, if the function object is func square(x) [parent=Global], the intrinsic name is square.
    - The parent frame ([parent=Global])
4. Bind the formal parameters to the argument values obtained in step 2 (e.g. bind x to 3).
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.

If a function does not have a return value, it implicitly returns None. In that case, the “Return value" box should contain None.

Let's put it all together! Draw an environment diagram for the following code.

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

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