Worksheet 1:

Environment Diagrams & Controls

Controls:

Conditional statements can help us decide which lines of code to execute in different scenarios.

Eg:

If it is morning, Larry will eat Breakfast

If it is afternoon, Larry will eat Lunch

If it is evening, Larry will eat Dinner

How can we write this scenario in Python? Using if-elif-else syntax and conditional expressions.

```
if <conditional expression>:
      <suite of statements>
elif <conditional expression>:
      <suite of statements>
else:
      <suite of statements>
```

A conditional expression is an expression that evaluates to a True or False value. In addition to any Boolean value, we can use `==`, `!=` to see if two values are the same, and `>`, `<`, `>=`, `<=` to see which value is larger or smaller. These evaluate to True or False as well.

True-y values: `True, a non-zero integer`

False-y values: `False, 0, None, "", []`

If a conditional expression evaluates to True, then the corresponding suite will be executed. If none of them are executed, it will automatically call the suite after the `else` case.

We can also use **Boolean operators** to manipulate to manipulate the above Boolean values. Eg. `and, or, not`

and: Will STOP and RETURN the first False value. If all values are True, it will return the LAST expression

Eg:

```
>>> 1 and 0
 0
1 is evaluated, then 0 is evaluated. Since 0
is the first False-y value, it is returned.
>>> 0 and 1
0
0 is evaluated. Since 0 is False, it is returned. 1
is not evaluated.
>>> 'soda' and 'cory'
'cory'
'soda' is evaluated, then 'cory' is evaluated.
Since both are truthy values, the last value is
returned.
```

or: Will STOP and RETURN the first True value. If all values are False, it will return the LAST expression.

Eg:

>>> 1 or 0

```
1
```

```
1 is evaluated. Since 1 is True, it is returned. 0
is not evaluated.
```

>>> 0 and 1

```
1
```

```
0 is evaluated, then 1 is evaluated. Since 1
is the first Truthy value, it is returned.
```

>>> None and 0

```
0
```

```
None is evaluated, then 0 is evaluated. Since both
are false-y values, the last value is returned.
```

not: Returns the opposite of whatever is afterwards

Eg:

>>> not 'soda'

```
False
```

```
Since 'soda' is a Truthy value, and since the
opposite of True is False, False is returned.
```

So can we write a control structure for Larry's routine?

```
if morning == True:
        print('eat breakfast')
elif afternoon == True:
        print('eat lunch')
elif evening == True:
        print('eat dinner')
```

First, we check if it is morning. If it is (morning == True evaluates to True), then Larry can eat breakfast. If it is not (morning == True evaluates to False), then we go to the next conditional statement, etc.

Questions:

Control Review:

```
> True AND True

> True AND False

> False AND True

> False AND False

> 1/0 AND True

> False AND 1/0

> True AND 1/0


> True OR True

> True OR False

> False OR True

> False OR False

> 1/0 OR True
```

```
> False OR 1/0

> True OR 1/0


> not False

> False or not False
```

More Questions:

```
>  False or False or True or False

>  'I' and 'love' and 'soda' or 'hall'

>  'Meesa' and 'has' and 0 or 'midichlorians'

>  1/0 or 2

>  2 or 1/0

>  'that' and 'that' and 'that' or 0

>  not 'that' or 'this' and not 0

>  None or not 1 and not None
```

Code Writing Questions:

Write a function that returns False for all negative numbers, but True for all positive numbers (including 0).

```
def false_positive(number):
        """
        >>> false_positive(90)
                True
        >>> false_positive(-61)
                False
        >>> false_positive(0)
                True
    """
```

```
        if _______________

                _______________

        elif _______________

                _________________
```

Remember Larry? Well he's decided to eat less. He will now only eat a meal if he is hungry. (Don't be like Larry. Always eat all three meals every day for a healthy life!) Implement a function that prints eat _____ for each meal. If Larry is not hungry, then it should print 'not hungry'.

```
def time_to_diet(time, hungry):
        """
        >>> false_positive('morning', True)
                'eat breakfast'
        >>> false_positive('evening', False)
                'not hungry'
        >>> false_positive('afternoon', False)
                'not hungry'
    """
        if _______________:

                ___________________

        elif _____________:

                ____________________

        elif ___________________:

                _________________
        else:
                ____________________
```

Question: What does the function time_to_diet RETURN?

Answer: _____

Melanie will only wear her expensive perfume on a second date, but only if her first date had a rating (1-10) of 7 or better. There is one exception. If she goes on a date with Ben Affleck, she will wear her perfume on the first date. Return True if she should use her expensive perfume, and False otherwise. Try writing this with no skeleton code!

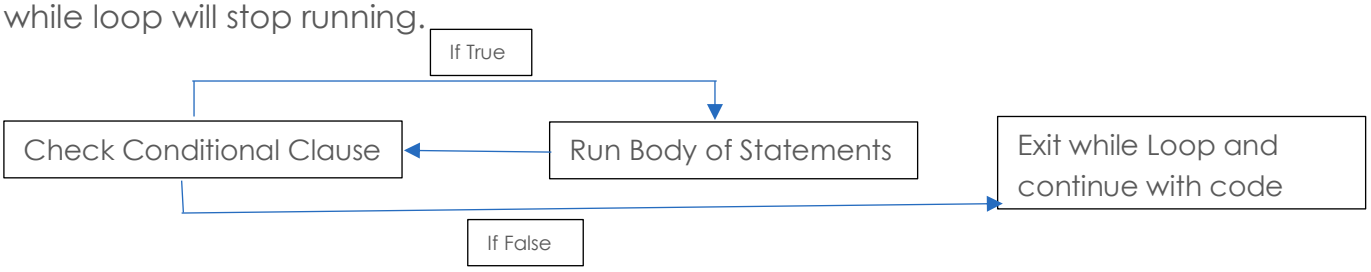
```
def expensive_perfume(date_number, first_date_rating, is_ben_affleck):
        """
        >>> false_positive(1, None, False)
                False
        >>> false_positive(1, None, True)
                True
        >>> false_positive(2, 4, False)
                False
        >>> false_positive(2, 9, False)
                True
        >>> false_positive(2, 2, True)
                False
    """
        ***YOUR CODE HERE***
```

While Loops

What if we want to repeat multiple things? For this, we can use a while loop.

```
while <conditional clause>:
        <body of statements>
```

The while loops will first check the conditional clause. If it is TRUE, it will run the body of statements. Then, it will check the conditional clause again, to see if it should re-run the body of statements. If the conditional clause is ever FALSE, the while loop will stop running.



Be Careful! While Loops can create infinite loops that never end!

Eg.

```
while True:
        print("Dormammu, I've come to bargain")
```

Here, the conditional clause is always True. Therefore, it will always print, as the conditional never changes.

Eg.

```
x = 0
while x < 5:
        print("Dormammu, I've come to bargain")
        x = x+1
print("Dr. Strange, I've had enough")
```

This code will print "Dormammu, I've come to bargain" 5 times. Each time it prints, it also increases x by 1. When x becomes 5, x<5 would return False, thus ending the while loop, and also Dormammu's misery.

Questions:

Implement a factorial function using a while loop.

```
def factorial(n):
    """
    >>> factorial(0)
    0
    >>> factorial(1)
    1
    >>> factorial(2)
    2
    >>> factorial(3)
    6
    >>> factorial(5)
    120
    """
    _________, _________ = n, ________

    while __________:
```

```
            _________, _________ = ________, _________
        return ____________
```

Environment Diagrams:

Drawing environment diagrams is an extremely powerful tactic to understand how a certain function works. It visually shows which variables are bound to what. You can use tutor.cs61a.org to create environment diagrams for any code you want to visualize.

```
def titanic(n):
        name = 'jack'
        n = n+5
        k = n
        n = 1997
        return "I'm the king of the world"
titanic(5)
```

Let's analyze this function.

First, we have assignment statements. We must first evaluate the name, and the value it is being bound to.

> name = 'jack'

We assign the String 'jack' to name

> n = n+5

We assign n to n+5

> k = n

We assign k to our new n value

> n = 1997

We re-assign n to 1997 {The year the movie *Titanic* came out}

We also have a def statement. There are certain aspects of a def statement that are important:

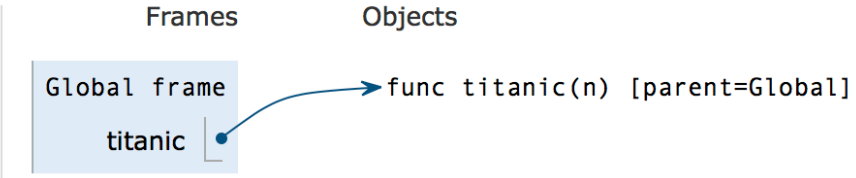
- The name (titanic)
- Any variables being passed into the function (n)
- The parent frame (Global) {We will expand on why this is when we cover Higher Order Functions}

Lastly, we have a function call

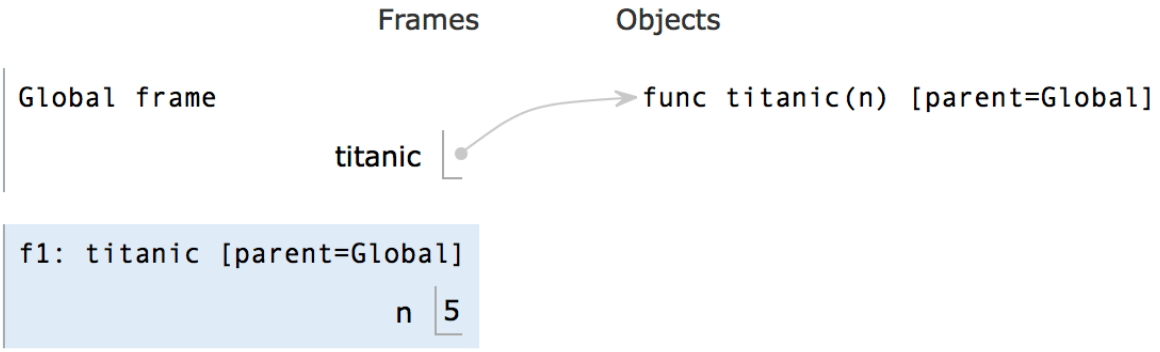
> titanic(5)

Here, we must take into account WHERE the function is being called, and evaluate the operator and then the operand(s).

Let's make an environment diagram now!



Here, we are declaring the function 'titanic', since our first line is a def statement. We DO NOT need to look inside the function, as we don't need that information until we CALL the function.



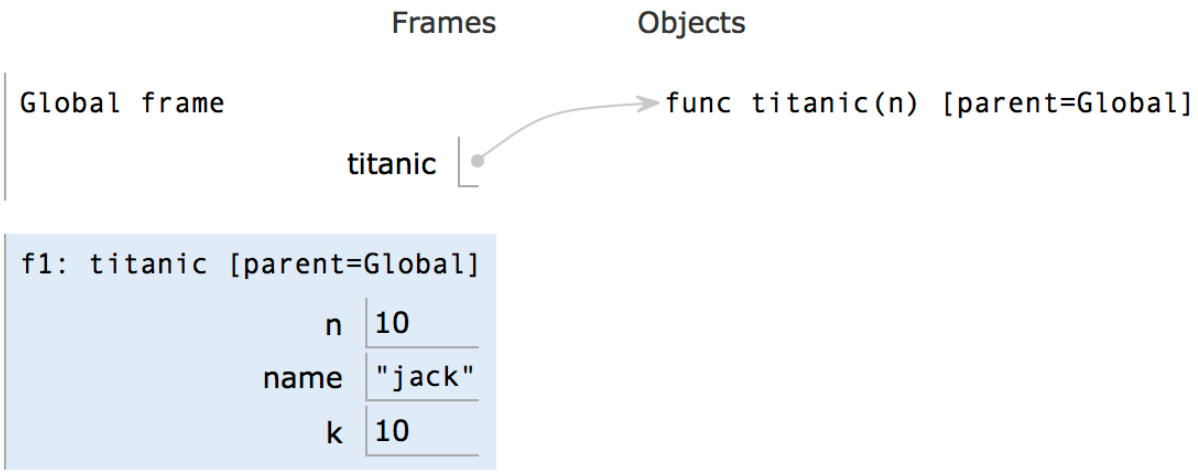
Next, we call the function at the line 'titanic(5)'. Since we have a function call, we create a new frame called 'f1'.

We evaluate the operator (titanic) and place that as the name of our function.

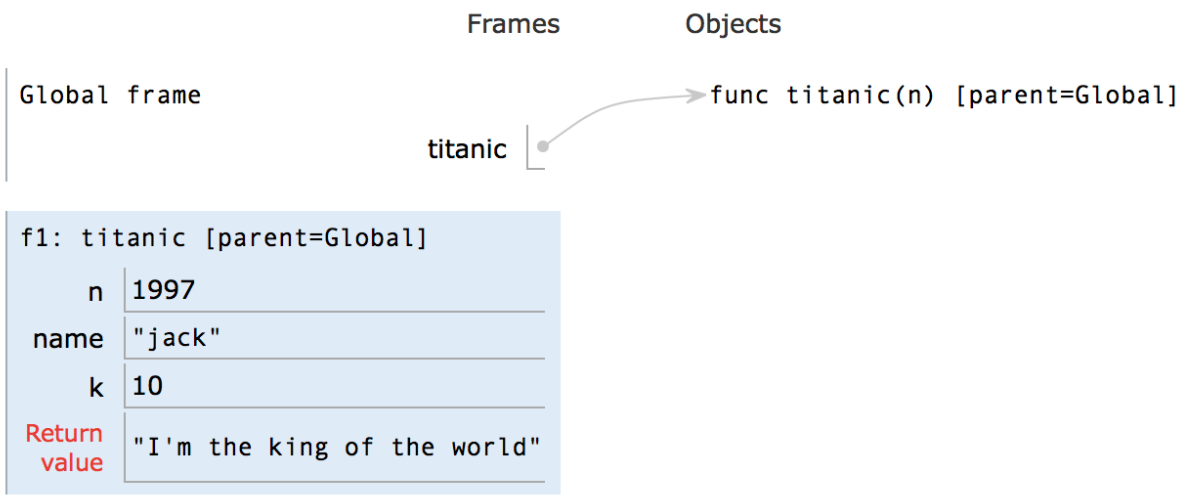
We use the information in the global frame to determine what titanic is bound to. We see the parent frame is Global, so we add that information.

Lastly, we evaluate our operand (5), and assign that to the operand of titanic 'n'.

We have just created our first frame!



Next, we have our assignment statements. We assign the String 'jack' to the variable name. We then reassign n to 10, and create a new variable k, and assign it to 10.



We then reassign n to 1997, and then finally we get to our return value. We place this in the box "Return Value", and this is the value that the function call returns.

This all may seem a little complicated, but it becomes easier with practice. Use this link for a more interactive step by step of the above example.

https://goo.gl/ToAhzs

Let's do some practice now!

Questions:

Draw the environment diagram for the following:

```
def gold_finger(name):
        new_name = 'paul'
        name = new_name
        n = 1964
        return "hilfinger"
gold_finger('bond')
```

```
k = 5
def i_love_math(m):
        m = m%5
        a = m+10
        t = a // 2
        h = t*4
        return m+a+t+h
i_love_math(k-2)
```