1 Higher Order Functions

A higher order function (HOF) is a function that manipulates other functions by taking in functions as arguments, returning a function, or both.

1.1 Functions as Arguments

One way a higher order function can exploit other functions is by taking functions as input. Consider this higher order function called negate.

```python
def negate(f, x):
    return -f(x)
```
	negate takes in a function \( f \) and a number \( x \). It doesn’t care what exactly \( f \) does, as long as \( f \) takes in a number and returns a number. Its job is simple: call \( f \) on \( x \) and return the negation of that value.
1.2 Questions

1. Here are some possible functions that can be passed through as \( f \).
   
   ```python
   def square(n):
       return n * n
   
   def double(n):
       return 2 * n
   
   What will the following Python statements output?
   ```
   ```
   >>> negate(square, 5)
   >>> negate(double, -19)
   >>> negate(double, negate(square, -4))
   ```

2. Implement a function `keep_ints`, which takes in a function `cond` and a number \( n \), and only prints a number from 1 to \( n \) if calling `cond` on that number returns `True`:
   ```python
   def keep_ints(cond, n):
       """Print out all integers 1..i..n where cond(i) is true
       >>> def is_even(x):
       ...     # Even numbers have remainder 0 when divided by 2.
       ...     return x % 2 == 0
       >>> keep_ints(is_even, 5)
       2
       4
       """
   ```
1.3 Functions as Return Values

Often, we will need to write a function that returns another function. One way to do this is to define a function inside of a function:

```python
def outer(x):
    def inner(y):
        ...
    return inner
```

The return value of `outer` is the function `inner`. This is a case of a function returning a function. In this example, `inner` is defined inside of `outer`. Although this is a common pattern, we can also define `inner` outside of `outer` and still use the same `return` statement.

```python
def inner(y):
    ...

def outer(x):
    return inner
```

1.4 Questions

1. Use this definition of `outer` to fill in what Python would print when the following lines are evaluated.

```python
def outer(n):
    def inner(m):
        return n - m
    return inner

>>> outer(61)
>>> f = outer(10)
>>> f(4)
>>> outer(5)(4)
```
2. Implement a function `keep_ints` like before, but now it takes in a number `n` and returns a function that has one parameter `cond`. The returned function prints out all numbers from `1..i..n` where calling `cond(i)` returns `True`.

```python
def keep_ints(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
>>> keep_ints(5)(is_even)
2
4
"""
```
1. Draw the environment diagram for evaluating the following code
```python
def f(x):
    return y + x
y = 10
f(8)
```

2. Draw the environment diagram for evaluating the following code
```python
def dessef(a, b):
c = a + b
b = b + 1
b = 6
dessef(b, 4)
```
3. Draw the environment diagram for evaluating the following code

```python
def foo(x, y):
    foo = bar
    return foo(bar(x, x), y)

def bar(z, x):
    return z + y

y = 5
foo(1, 2)
```
4. Draw the environment diagram for evaluating the following code

```python
def spain(japan, iran):
    def world(cup, egypt):
        return japan-poland
    return iran(world(iran, poland))

def saudi(arabia):
    return japan + 3

japan, poland = 3, 7
spain(poland+1, saudi)
```
5. Draw the environment diagram for evaluating the following code

cap = 9
hulk = 3

def marvel(cap, thor, avengers):
    marvel = avengers
    iron = hulk + cap
    if thor > cap:
        def marvel(cap, thor, avengers):
            return iron
    else:
        iron = hulk
    return marvel(thor, cap, marvel)

def iron(man):
    hulk = cap - 1
    return hulk

marvel(cap, iron(3), marvel)