COMP 333
Fall 2020

Functions and Higher-Order Functions in Swift

1.a.) Define a function named `add` which takes the `Int` parameters `a` and `b` and adds them together, returning the result of the addition. The caller of `add` should need to provide the labels `a` and `b`.

```
func add(a: Int, b: Int) -> Int {
  return a + b;
}
```

1.b.) Call `add` with parameters 2 and 3 (hint: this won't be the same as Java).

```
add(a: 2, b: 3)
```

2.a.) Define a function named `sub` which takes two `Int` parameters, subtracts the second from the first, and returns the result of the subtraction. The caller of `sub` should **not** need to provide any labels.

```
func sub(_ first: Int, _ second: Int) -> Int {
  return first - second;
}
```

2.b.) Call `sub` with parameters 4 and 5. (hint, this will be the same as Java).

```
sub(4, 5)
```

3.a.) Define a function named `callsFunc` which calls a passed function with a given parameter, returning the result of the call. `callMe` should have the following signature:

```
func callsFunc(f: (Int) -> Int, i: Int) -> Int
```

```
func callsFunc(f: (Int) -> Int, i: Int) -> Int {
  return f(i)
}
```

3.b.) Call `callsFunc` with the following parameters:

- A higher-order function that adds 1 to its parameter and returns the result
- The integer 5

```
callsFunc(f: { x in x + 1 }, 5)
```

4.) Define a function `indirectIf` which takes a `Bool` and two functions. If the `Bool` is `true`, it calls the first function, returning its result. If the `Bool` is `false`, it calls the second function, returning its result. Example calls are shown below (you should be able to determine indirectIf's signature from these):

```
// returns 2
indirectIf(true, ifTrue: { 1 + 1 }, ifFalse: { 2 + 2 })

// returns 8
indirectIf(false, ifTrue: { 3+3 }, ifFalse: { 4 + 4 })
```

```
func indirectIf(_ c: Bool,
                ifTrue: () -> Int,
                ifFalse: () -> Int) -> Int {
  if c {
    return ifTrue();
  } else {
    return ifFalse();
  }
}
```