# 15-112, Week 1, Conditionals Practice

See how many of these problems you and a partner can solve during today's lab.

For each problem, you write the requested function *and* testcases for it. See `week01/example.py` in the demo code repository for an example of organizing this.

**Rules for every problem below:** no strings, no loops, no lists, and no recursion. You have conditionals, arithmetic, and boolean logic — that's it.

---

### 1. isFactor(f, n)

Write the function `isFactor(f, n)` that takes two `int` values `f` and `n`, and returns `True` if `f` is a factor of `n`, and `False` otherwise. Note that every integer is a factor of `0`.

---

### 2. isMultiple(m, n)

Write the function `isMultiple(m, n)` that takes two `int` values `m` and `n`, and returns `True` if `m` is a multiple of `n`, and `False` otherwise. Note that `0` is a multiple of every integer other than itself. Make constructive use of the `isFactor` function you just wrote.

---

### 3. isLegalTriangle(s1, s2, s3)

Write the function `isLegalTriangle(s1, s2, s3)` that takes three `int` or `float` values representing the lengths of the sides of a triangle, and returns `True` if such a triangle exists, and `False` otherwise. Recall the triangle inequality: the sum of any two sides must be greater than the third side, and all three sides must be positive. *Hint: think about how to identify the longest side, and why that helps.*

---

### 4. rectanglesOverlap(left1, top1, width1, height1, left2, top2, width2, height2)

A rectangle can be described by its left, top, width, and height. This function takes two rectangles described this way, and returns `True` if the rectangles overlap at all (even just at a single point), and `False` otherwise. Use the usual computer-graphics coordinate system, where `(0, 0)` is the top-left corner of the screen, x increases to the right, and y increases *downward*.

---

### 5. integerSquareRoot(v)

We define the integer-square-root of a non-negative integer `N` to be the largest integer `M` such that `M**2 <= N`. So the integer-square-root of `99` is `9`, because `9**2 = 81 <= 99`, but `10**2 = 100` is not `<= 99`. Write the function `integerSquareRoot(v)` that takes an arbitrary value `v` (which may not even be an integer), and if `v` is a non-negative integer, returns its integer-square-root. Otherwise, the function returns `None`.

---

### 6. isSmallTidy(n)

We'll say a value `n` is a *small tidy* (coined term) if it is an integer with precisely three digits whose digits are either strictly increasing or strictly decreasing in order. For example, `123`, `321`, and `-941` are all small tidy numbers. Write the function `isSmallTidy(n)` that takes a value `n`, which may or may not be an integer, and returns `True` if `n` is a small tidy number, and `False` otherwise. Don't crash if `n` isn't an integer!

---

### 7. isSmall42ish(n)

We'll say a value `n` is a *small 42ish* (coined term) if it is an integer with precisely four digits that contains at least one pair of consecutive digits forming `42`. For example, `4212`, `5042`, and `-4242` are all small 42ish numbers. Write the function `isSmall42ish(n)` that takes a value `n`, which may or may not be an integer, and returns `True` if `n` is a small 42ish number, and `False` otherwise. Don't crash if `n` isn't an integer!

---

### 8. isSmallMountain(n) — challenge problem

We'll say a value `n` is a *small mountain* (coined term) if it is an integer with precisely five digits (ignoring a leading `-` sign) whose digits strictly increase up to some peak digit and then strictly decrease afterward — with the peak being neither the first nor the last digit. For example, `13531` is a small mountain (`1 < 3 < 5`, then `5 > 3 > 1`), and so is `-29640` (`2 < 9`, then `9 > 6 > 4 > 0`). However, `12345` is not a small mountain (it never decreases), and `54321` is not (it never increases). Write the function `isSmallMountain(n)` that takes a value `n`, which may or may not be an integer, and returns `True` if `n` is a small mountain number, and `False` otherwise. Don't crash if `n` isn't an integer!