Programming Exercises
Last updated: March 23rd, 2016

Writing a complete Processing sketch is like playing Beethoven's Moonlight Sonata through from start to finish on the piano. There are twists and turns, and the process requires a balance of endurance and creativity. Most importantly, it relies on a degree of skill with the basic techniques of piano playing. Before you can play a complex piece you must first master basic skills like playing scales, arpeggios, chords and inversions. You must practice playing at different speeds and volumes.

These exercises are the Processing equivalent of practicing your scales. They are very small, self-contained programming questions that force you to think about different aspects of coding in isolation.

Unless the question very specifically tells you otherwise, you should assume that the function takes values directly as parameters and returns a value as its answer. That is, you shouldn't try to use `println()` or `createWriter()` to output information, and you shouldn't try to load information from external files via `loadStrings()`. Questions marked with two asterisks (**) are a bit more challenging.

Functions

1. Write a function called `doNothing()` that takes no arguments as input, produces no values as output, and does nothing.
2. Write a function called `identity()` that takes a single integer as an argument and returns that integer.
3. Write a function `plus()` that takes three floating-point numbers as input and returns their sum.
4. Write a function called `secondsElapsed()` that takes no arguments and returns a floating-point value equal to the number of seconds that have elapsed since the sketch was started. Use the built-in function `millis()` as a helper.
5. Write a function called `numberOfPixels()` that tells you the number of pixels in the sketch window.
6. Write a function called `imagePixels()` that takes an image as input and returns the number of pixels in the image.
7. Write a function called `isSix()` that takes a single integer as input and returns a boolean value telling you whether the passed-in integer was the number 6. Write it once using an `if` statement, and once without an `if` statement.
8. Write a function called `circle()` that takes three floating-point numbers as input representing the *x* and *y* coordinates of the centre of a circle and a radius for that circle, and draws the circle.
9. Write a function called `firstOfSecond()` that takes a string as input containing at least two words separated by spaces, and returns the first letter of the second word. The return type should be `char`.

10. Write a function called `xor()` that takes two boolean values as input and returns a boolean that's true if *exactly one* of the two inputs is true (but not both). (**) Write it without using an `if` statement.
11. Write a function `endsWithPunctuation()` that takes a string as input and returns a boolean telling you whether the last character in the string is a punctuation character. Write a simple version that assumes the input string is non-empty, and then write a slightly more complicated version that checks whether the string is empty and in that case answers `false`.
12. Write a function `bmi()` that takes two floating-point numbers as input representing your height in metres and your mass in kilograms and returns your Body Mass Index (you may have to look up for the formula for BMI).
13. Write a function `fahrenheit()` that takes the current temperature in degrees celsius as input and returns the temperature in degrees fahrenheit.
14. Write a function `isPythagorean()` that takes three positive integers as input and tells you whether the square of any of the three is equal to the sum of the squares of the other two.

Arrays

1. Write a function `countElements()` that takes an array of integers as input and returns how many integers are in the array. Do not use a loop.
2. Write a function `hasZero()` that takes an array of integers as input and returns a boolean telling you whether the array contains any zeros. Do not look at any more array elements than you have to in order to determine the answer.
3. Write a function `contains()` that takes two parameters as input: an array of integers and a single integer. It returns a boolean that tells you whether the passed-in integer occurs at any point in the array.
4. Write a function `countNegative()` that takes an array of integers as input and returns how many elements in the array are less than zero.
5. Write a function `countOccurrences()` that takes two parameters as input: an array of integers and a single integer. It returns an integer that tells you how many times the passed-in value occurs at any point in the array.
6. Write a function `largestElement()` that takes an array of integers as input and returns the largest integer in the array. Assume the array has length at least one.
7. Write a function `average()` that takes an array of integers as input and returns a floating-point number representing the average of the elements in the array. Assume the array has length at least one.
8. Write a function `drawPointsInterleaved()` that takes an array of floating-point numbers as input containing an alternating sequence of *x* and *y* coordinates and draws single points at all of those positions. That is, array positions 0 and 1 represent a point, positions 2 and 3 represent a point, and so on.
9. Write a function `drawPointsParallel()` that takes two arrays of floating-point numbers as input. The first array is a sequence of *x* coordinates and the second is a sequence of *y* coordinates, and the arrays have the same length. The function draws a point at each corresponding *x*, *y* pair.

10. Repeat the question above, but do not assume that the two arrays have the same length. When either array runs out, stop drawing points.
11. Write a function `squares()` that takes a single positive integer as input and returns an array of that length, containing a sequence of perfect squares starting from 0. For example, `squares( 5 )` would return the array `{ 0, 1, 4, 9, 16 }`.
12. Write a function `range()` that takes two integers as input and returns an array starting from the first integer and ending with the number before the second integer. For example, `range( 3, 9 )` would return the array `{ 3, 4, 5, 6, 7, 8 }`.
13. Extend the previous function so that the second number is permitted to be smaller than the first number, in which case the array counts down instead of up.
14. Write a function `doubler()` that takes an array of integers as input and returns a new array twice as long in which each element of the original array is repeated. That is, if you pass in the array `{ 1, 2, 3 }` you would get back the array `{ 1, 1, 2, 2, 3, 3 }`.
15. Write a function `sums()` that takes two arrays of integers as input, which are assumed to have the same length, and returns a new array containing the sums of the corresponding elements of the original arrays.
16. Write a function `isSorted()` that takes an array of integers as input and returns boolean telling you if the elements of the array are in increasing order.
17. Write a function `partialSums()` that takes an array of integers as input and returns an array of integers where each element is the sum of all the numbers up to that point in the original array. That is, if you pass in `{ 1, 2, 3, 4, 5 }` as input you would get `{ 1, 3, 6, 10, 15 }` as output.
18. (**) Write a function `hasDuplicate()` that takes an array of integers as input and returns a boolean that tells you whether any element in the array occurs multiple times.
19. Write a function `swap()` that takes three values as input: an array of integers, and two additional integers which act as indices into the array. The function should change the array in-place so that the elements at the given indices are swapped. For example, if `a` is the array `{ 1, 4, 9, 16, 25 }`, then after calling `swap( a, 2, 4 )` the array should contain `{ 1, 4, 25, 16, 9 }`.
20. Write a function `scramble()` that takes an array of integers as input and randomly rearranges the elements of the array in-place. The function should *not* return a value, but should move elements around in the original array. (Hint: this can be done by performing a large number of random swaps using the function above as a helper.)
21. Write a function `repeat()` that takes a string and a non-negative integer as input and returns an array of strings with the string repeated the number of times given by the integer. For example, `repeat( "pie", 4 )` would return the array `{ "pie", "pie", "pie", "pie", "pie" }`.
22. Write a function `replaceAll()` that takes three values as input: an array of integers, and separate integers `a` and `b`. It returns a brand new array that's identical to the one passed in except that every occurrence of `a` has been replaces by `b`.
23. (**) Write a function `removeAll()` that takes an array of integers and a separate integer `a`. It returns a brand new array that's similar to the one passed in, except that every occurrence of `a` has been removed. In general, if the input array contains any occurrences of `a` then the result will be a shorter array.

The great trinity

1. Write your own version of the built-in function `lerp()` as a one-liner using `map()`.
2. Write your own version of the built-in function `norm()` as a one-liner using `map()`.
3. Write your own version of the built-in function `map()` as a one-liner using `lerp()` and `norm()`.
4. Write a function called `halfsies()` that takes two floating-point numbers as input and returns their average. Write it as a one-liner using `lerp()`.
5. (**) Write the function `map()` from scratch, without using `lerp()` or `norm()`.

Geometry

1. Write a function `pointInCircle()` that takes five floating-point numbers as input: the *x* and *y* coordinates of a point, and the *x*, *y*, and *r* values that define the centre and radius of a circle. The function returns a boolean telling you whether the point lies anywhere within the circle.
2. Write a function `pointOnCircle()` that takes the same inputs as above, but tells you if the point lies on the boundary of the circle itself instead of inside it. The point will almost never lie *exactly* on the boundary, so you'll need to check whether it's within some small margin of error from the boundary.
3. Write a function `trianglePerimeter()` that takes six floating-point numbers as input, representing the *x* and *y* coordinates of the corners of a triangle, and returns the perimeter of the triangle (the sum of the lengths of the sides).
4. Write a function `triangleArea()` that takes six floating-point numbers as above, and returns the area of the triangle represented by those coordinates. There are a few ways to do this. For example, look up "Heron's formula" for a convenient method.
5. (**) Write a function `perimeter()` that takes an array of `PVector`s as input, representing two-dimensional points, and returns the total length of the closed path defined by the sequence of points. Assume that each point is connected by a line to the next point i the array, and that the array is "circular": the last point connects back to the first one.

Strings

1. Write a function mcFace() that takes a string as input and returns a new string in which the passed-in string has been inserted to make a phrase like `"Boaty McBoatFace"`. That is, any string `"XXX"` you pass in takes the place of `"Boat"`, yielding `"XXX McXXXFace"`.
2. Write a function `reverse()` that takes a single string as input and returns a new string containing all the characters of the original string in reverse order.
3. Write a function `reverseWords()` that takes a single string as input and returns a new string containing all the words of the original string in reverse order. For example, `reverseWords( "It was the best of times" )` should return `"times of best the was It"`.

4. Write a function `reverseEachWord()` that takes a single string as input and returns a new string in which each individual word has been reversed. For example, `reverseEachWord( "It was the best of times" )` should return `"tI saw eht tseb fo semit"`.
5. Write a function `capitalize()` that takes a string as input and capitalizes the first letter of each word in the string. For example, `capitalize( "galactic president superstar mcawesomeville" )` would return `"Galactic President Superstar Mcawesomeville"`.
6. Write a function `isPalindrome()` that takes a string as input and returns a boolean that tells you whether the string is a palindrome (i.e., if it's the same when written backwards).
7.
8. (**) Write a fancier version of the previous function that ignores capitalization, punctuation and whitespace. In this case, `isPalindrome( "Madam, I'm Adam." )` would return `true`. You may need to write a helper function or two.
9. (**) Write a function `jumble()` that takes a string as input containing a single word and returns a new string in which all of the characters have been randomly rearranged. For example, `jumble( "wastrel" )` might return `"selrwat"`. (Hint: this can be done using the `scramble()` function from the Arrays section.)

Recursion

1. (**) Write three simple functions that take single integers as input: `inc()`, which returns one more than the passed in number, `dec()`, which returns one less, and `isZero()`, which returns a boolean telling you whether the input is zero. Now write a recursive function calls `plus()` that takes two non-negative integers *a* and *b* as input and returns *a* + *b*. *Do not use the + operation*. Instead, use recursion, with a base case of *b* = 0.
2. Write a recursive function `fact()` that takes an integer *n* as input and returns *n*!. This function is defined with a base case of 0! = 1 and a recursive case of *n*! = *n**(*n*-1)!. Now write the same function using a loop, not recursion.
3. Write a recursive function `fib()` that takes an integer *n* as input and returns F_n, the *n*th Fibonacci number. These numbers are defined with two base cases, $F_0 = 1$ and $F_1 = 1$, and a recursive case $F_n = F_{n-2} + F_{n-1}$. (**) Now write the same function using a loop, not recursion.

Classes

1. Write a class `Tao` that has no fields and no methods.
2. Write a class `Counter` that has a single integer field. Give it a method `increment()` that adds one to the field.
3. Write a class `Circle` that describes a single circle in the plane. The class should have three fields: the *x* and *y* coordinates of the centre, and a radius. Give the class a reasonable constructor.
4. Write a class `Rectangle` that describes a single rectangle in the plane. The class should four fields: the *x* and *y* coordinates of the top-left corner of the rectangle

(assuming that we're in a coordinate system where *y* grows downward) and the rectangle's width and height. Give the class a reasonable constructor.

5. Give the `Rectangle` class an `area()` method that returns the area of the rectangle.
6. Repeat the previous exercise for the `Circle` class. The area of a circle is πr^2, where *r* is the circle's area.
7. Give the `Circle` class a method `contains()` that takes two floating-point numbers as input, representing the *x* and *y* coordinates of a point, and returns a boolean indicating whether the point lies within the circle.
8. Repeat the question above for the `Rectangle` class.
9. (**) Write a second constructor for the `Rectangle` class, which takes a `Circle` as input and initializes itself to be the smallest `Rectangle` that contains that circle.
10. (**) Write a second version of the `contains()` method in the `Circle` class, which takes a `Rectangle` as input and returns a boolean indicating whether the rectangle is completely contained within the circle.
11. Add an `engulf()` method to the `Rectangle` class, which takes two floating-point values as input representing the *x* and *y* coordinates of a point, and modifies the rectangle so that it grows to contain the new point as well as its former self. The method should change the rectangle in-place and should not return a value.
12. Give the `Rectangle` class a method `union()` that takes a single rectangle as input and outputs a new rectangle. The new rectangle should be the smallest one that contains both the initial rectangle that received the message, and the rectangle that was passed in as a parameter. The method should produce a brand new rectangle as its return value without modifying either of the rectangles that were used in the computation.