# Macintosh HD:Users:gloria:Desktop:TRABAJOS:La Florida:@ Florida AF :Florida - Marcas AF:300dpi:Florida - Marca 300dpi-Universitària_c.png GUIÓN DE ACTIVIDAD

# Programación

## Actividad: Banco de ejercicios sobre funciones

## Objetivos

* *Encapsular funcionalidad y secuencia de instrucciones en funciones.*
* *Importar módulos de funciones externas*
* *Llamar funciones externas para realizar tareas*

## Temporalización

###### Aproximadamente 20 horas

## Actividades

1. Crea una función llamada isEven para determinar si un número x dado por parámetro es par. La función deberá devolver cierto en caso de que el número sea par, y falso en caso contrario.
2. Crea una función llamada isOdd para determinar si un número x dado por parámetro es impar. Emplea para ello en la lógica de la función llamada(s) a la función isEven programada anteriormente. La función deberá devolver cierto en caso de que el número sea impar, y falso en caso contrario.
3. Crea una función llamada isPalindrome para determinar si una cadena de texto inputString es un palíndromo o no. Una palabra es palíndroma cuando ésta se lee igual de izquierda a derecha, que de derecha a izquierda. La función deberá devolver cierto en caso de que la palabra sea palíndroma, y falso en caso contrario
4. Crea una función llamada isPrime para determinar si un número x dado por parámetro es primo o no. Recordar que un número es primo cuando únicamente es divisible por 1 y sí mismo. La función deberá devolver cierto en caso de que el número sea primo, y falso en caso contrario.
5. Crea una función llamada isLeapYear que tome por parámetro un año llamado year y devuelva cierto en caso de que el año sea bisiesto y falso en caso contrario. Puedes ayudarte de la lógica del ejercicio programada en las actividad de estructuras condicionales y encapsularla en una función.
6. Crea una función llamada average que tome por parámetro un array llamado dataArray y calcule la media de los valores almacenados en el array. La función deberá devolver la media de dicho vector y undefined en caso de que el array no tenga elementos.
7. Crea una función llamada greaterThanAverage que tome un parámetro x de tipo numérico, y un array llamado dataArray. La función deberá devolver cierto en caso de que el valor x sea mayor que la media del vector, y falso en caso contrario. Apóyate en la función desarrollada en la actividad anterior.
8. Crea una función llamada linearSearch que tome un parámetro x de tipo numérico, y un array llamado dataArray. La función deberá devolver cierto en caso de que el valor x se encuentre en dataArray, y falso en caso contrario. No se puede emplear el método/instrucción include definido para los arrays.
9. Crea una función llamada circumferenceArea que tome como parámetro el radio de la circunferencia radius y devuelva el área de la circunferencia.
10. Crea una función llamada quadraticSolver que tome como parámetro los términos a, b, y c de una ecuación de segundo grado (a\*x2 + b\*x + c) y devuelva las soluciones de la misma (aquellos valores que la hacen cero). El resultado será retornado en un array con dos posiciones, donde la primera posición es la solución de la ecuación cuando empleamos el término +, y la segundo posición es la solución de la ecuación cuando empleamos el término -. Si no existe solución para alguno de los dos casos, se deberá devolver NaN (not a number) en la posición que toque. Puedes apoyarte en el código desarrollado para la primera semana de contenidos.
11. Crea una función llamada programmingMarks que tome por parámetro la nota del proyecto de programación projectMarks, la nota de las actividades prácticas en un array activitiesMarks, y la nota del examen examMarks, y devuelva la nota del alumno para la asignatura de programación. Se deberá devolver la nota en caso de que el estudiante apruebe, la nota en caso de que el estudiante haya suspendido con una media ponderada de menos de 5, y una nota de 4.99 en el caso de que el estudiante haya suspendido con una media ponderada mayor de 5. Puedes apoyarte en las funciones desarrolladas anteriormente para implementar esta función.
12. Crea una función llamada everyElementGreaterThan que tome por parámetro un número x, y un array de datos dataArray. La función devolverá cierto en caso de que TODOS los elementos sean mayores que x, y falso en caso contrario.
13. Crea una función llamada withoutVowels que tome por parámetro una cadena de texto llamada text, y retorne una nueva cadena de texto donde todas las vocales han sido eliminadas.
14. Crea una función llamada hasRepeatedWords que tome por parámetro una cadena de texto llamada word y un array de palabras llamado wordsArray. La función deberá devolver cierto si al menos existe una palabra repetida en el array, y falso en caso contrario. No emplear el método includes de los arrays ni otras estructuras de datos que no sean arrays.
15. Crea una función llamada diceThrow que tome por parámetro un número de caras sides y simule la tirada de un dado con ese número de caras. Se deberá devolver el resultado de la tirada.
16. Crea una función llamada createRPGCharacter que creará los atributos de un personaje de rol: strength, speed, vitality, dexterity y magic. La función tomará como entrada una cadena de texto llamada race y que especificará la raza del personaje. La función devolverá en un array los valores de strength (STR), speed (SPD), vitality (VIT), dexterity (DEX), y magic (MAG) creados para ese personaje (en ese mismo orden). Por defecto, todos los atributos de personaje se determinarán con la tirada de un dado de 20 caras. Dependiendo de la raza, la creación cambiará de la siguiente forma:
    * Human: STR: 8 + 2 dados de 6, SPD: 4 + 1 dado de 6, MAG: 1 dado de 10
    * Elven: STR: 6 + 2 dados de 4, VIT: 1 dado de 12, DEX 3 + 2 dados de 10
    * Dwarf: STR: 10 + 1 dado de 10, SPD: 2 dados de 8, Vitality: 10 + 1 dado de 8, MAG: 1 dado de 8
    * Mage: STR: 1 dado de 6, VIT: 1 dado de 10, MAG: 12 + 1 dado de 8.
17. Crea una función llamada getHealthPoints que tomará como parámetro un array representando a un personaje en nuestro juego de rol y devolverá los puntos de vida del personaje:
    * Human: STR + VIT + 10
    * Elven: STR + DEX + 5
    * Dwarf: STR + VIT + 15
    * Mage: MAG + VIT + 3
18. Crea una función llamada getDamage que tomará como parámetro un array representando a un personaje en nuestro juego de rol y devolverá el daño base de dicho personaje redondeado al entero inmediatamente superior:
    * Human: 1.5 \* STR + 0.4 \* DEX
    * Elven: STR + 1.2 \* DEX + 0.1 \* MAG
    * Dwarf: 2 \* STR
    * Mage: 2\*MAG + 0.1 \* DEX
19. Crea una función llamada getDefense que tomará como parámetro un array representando a un personaje en nuestro juego de rol y devolverá la defensa base de dicho personaje redondeado al entero inmediatamente superior:
    * Human: VIT + 0.2 \* SPD
    * Elven: 1.3 \* SPD + 0.5 \* VIT
    * Dwarf: 1.1 \* VIT + 1.1 \* STR
    * Mage: 0.8 \* MAG + 0.3 \* SPD
20. Crea una función llamada simulateBattle que simulará el combate de dos personajes del juego anterior. El primer parámetro será una cadena de texto firstRace representando la raza del primer personaje. El segundo parámetro será un array representando los atributos del primer personaje (firstCharacterStats). Tras ello tendremos como parámetro la raza del segundo personaje secondRace y el array representando los atributos del segundo personaje (secondCharacterStats). Los combates se desarrollan de la siguiente forma:
    * Los puntos de vida del personaje vienen determinados por la fórmula
    * Cada turno, comenzará atacando aquel personaje con mayor velocidad.
      1. El daño producido por un personaje a otro será igual a su daño base menos la defensa base del otro personaje. Esto resta los puntos de vida del personaje que recibe daño
      2. Tras ello, y siempre que le queden puntos de vida al personaje que recibió daño, éste atacará siguiendo la misma lógica que el anterior ataque.
    * El combate termina cuando uno de los dos personajes se queda sin puntos de vida.
    * La función devolverá un array con dos elementos: el número de turnos que ha llevado finalizar el combate, y un string que será ‘first’ en caso de que venza el primer personaje, y ‘second’ en caso de que venza el segundo.
    * Apóyate en las funciones programadas anteriormente para hacer este desarrollo mucho más fácil.
21. Crea una función llamada getClosestEnemyIndex que tome como parámetro un array de dos posiciones representando las posiciones x e y de un personaje llamado characterPosition, y un segundo parámetro que será un array de enemigos llamado enemyPositions donde cada valor introducido será un array con las posiciones x e y de un enemigo. La función deberá determinar el índice en el array del enemigo más cercano al personaje. En principio, emplea la distancia euclidea para determinar la distancia entre dos pares de coordenadas (<https://en.wikipedia.org/wiki/Euclidean_distance#Two_dimensions>). Puedes construirte una función llamada euclideanDistance que tome como parámetros dos coordenadas, y calcule la distancia euclidea entre ambas posiciones.
22. Crea una función llamada standardDeviation para calcular la desviación típica (<https://es.plusmaths.com/como-calcular-la-desviacion-tipica.html>) de un array de números llamado dataArray pasado por parámetro. Puedes apoyarte en la función average calculada anteriormente para el cálculo.
23. Crea una función llamada reverseWord que tome como parámetro una cadena de texto word, y devuelva una nueva cadena consistente en la inversión del orden de las letras de dicha palabra (el primero es el último, y el último el primero). No emplees la instrucción reverse ni instrucciones externas de inversión de cadenas.
24. Ahora crea una función llamada isPalindrome2 que tome como parámetro una cadena de texto word y determine si la palabra es un palíndromo. Para ello, esta vez apóyate en la función reverseWord construida en el ejercicio anterior.
25. Ahora escribe una función llamada swapWithMinimum que tome como parámetro una posición dentro de un array definida como index, y un array de números llamado dataArray. La función deberá modificar el array de forma que intercambie el valore almacenado en la posición apuntada por index con la posición del valor más pequeño encontrado entre index y el final del array.
    * Por ejemplo, si index = 1, y el array es [ 4, 1, 3, 7, 0 , 9], la función deberá intercambiar los valores de 1 y 0 en el array, quedando de la siguiente forma: [4, 0, 1, 3, 7, 1, 9]
    * Por ejemplo, si index = 4, y el array es [4, 1, 3, 7, 0, 9] la función no intercambiará el valor del 0 con otro al ser el mínimo comprendido entre la posición 4 del array y el final del mismo.
26. Apoyándote en la función anterior, construye una función llamada sortArray que ordene los valores numéricos de un array pasado por el parámetro dataArray de menor a mayor. (Pista: Descompon el problema de la ordenación de todo el array, en los problemas de encontrar el elemento a posicionar en cada una de las posiciones del array). En este caso la función no devolverá ningún valor sino que simplemente modificará el array de entrada.
27. Implementa una función llamada deleteAt que tome un array de elementos como entrada llamado dataArray, y una posición index en el array, y devuelva un nuevo array donde el valor contenido en la posición index ha sido borrado.
28. Implementa una función llamada isWithinRectangle que tome como entrada dos coordenadas topLeftCoordinate y bottomRightCoordinate representando respectivamente las coordenadas de la esquina superior izquierda e inferior derecha de un rectángulo, y otra tercera coordenada llamada coordinate. La función deberá devolver true en caso de que la tercera coodenada se encuentre contenida dentro del rectángulo.
29. Implementa una función llamada isValidDate que tome como parámetros un número de mes month, un número de día day, y determine si la fecha introducida por el usuario es válida o no. La función devolverá cierto si la fecha es válida y falso en caso contrario.