Ejercitación № 9: Listas

Cátedra Programación II Mayo 2018

1. Ejercicios con Listas

Desarrollar cada uno de los siguientes ejercicios de forma completa, detallando cada una de las etapas involucradas en la construcción de un programa, tal cual las presentamos en la receta. Para la etapa de testing, se deberá emplear el módulo `pytest`. Tener en cuenta que, deberá crear funciones que permitan el pasaje de argumentos para posteriormente testear el programa.

Ejercicio 1. Escriba una función `posicionesMultiplo` que tome una lista y, un número, y retorne la lista formada por los elementos que están en las posiciones múltiplos de ese número. Por ejemplo:

- `posicionesMultiplo([1,2,3,4,5,6,7],2)` retorna `[1,3,5,7]` y,
- `posicionesMultiplo([1,2,3,4,5,6,7],3)` da como resultado `[1,4,7]`.

Ejercicio 2. Escriba una función que tome una lista de números y devuelva la suma acumulada, es decir, una nueva lista donde el primer elemento es el mismo, el segundo elemento es la suma del primero con el segundo, el tercer elemento es la suma del resultado anterior con el siguiente elemento y así sucesivamente. Por ejemplo, la suma acumulada de `[1,2,3]` es `[1, 3, 6]`.

Ejercicio 3. Escriba una función llamada `elimina` que tome una lista, y elimine el primer y último elemento de la misma, creando una nueva lista con los elementos que no fueron eliminados.

Ejercicio 4. Escriba una función `ordenada` que toma una lista como parámetro y devuelva *True* si la lista está ordenada en orden ascendente y devuelva *False* en caso contrario. Por ejemplo,

- `ordenada([1, 2, 3])` retorna *True* y
- `ordenada(['b', 'a'])` retorna *False*.

Ejercicio 5. Escriba una función llamada `duplicado` que tome una lista y devuelva *True* si tiene algún elemento duplicado. La función no debe modificar la lista.

Ejercicio 6. Escriba una función llamada `eliminaDuplicados` que toma una lista y devuelva una nueva lista con los elementos únicos de la lista original. No tienen porque estar en el mismo orden.

Ejercicio 7. Escriba una función que tome una lista y retorne la cantidad de elementos distintos que tiene. Se recomienda usar la función anterior.

Ejercicio 8. **Búsqueda Dicotómica**

Para comprobar si una palabra está en una lista se puede utilizar el operador **in**, pero sería una búsqueda lenta, ya que busca a través de las palabras según el orden que están en la lista. Si la lista almacena las palabras en orden alfabético, podemos acelerar las cosas con una búsqueda dicotómica (también conocida como búsqueda binaria), que es similar a lo que se realiza, cuando se busca una palabra en el diccionario. Comenzamos por el centro y comprobamos si la palabra que buscamos está antes o después del centro. Si está antes, se busca sólo en la primera mitad, si está después se busca en la otra mitad de la lista. Con esto reduciremos el tiempo de búsqueda.

Se pide que implemente la función `busquedaDicotomica` que toma una lista de palabras ordenadas alfabéticamente y, una palabra a buscar y, retorna si la palabra está en la lista o no. Puede darse una implementación recursiva o iterativa.

2. Reveemos ejercicios de Racket usando listas y ciclos en Python

Reeutilizar la receta de los ejercicios resueltos en `Racket`, Práctica 5, para darles ahora una solución en `Python`. Cómo la solución recursiva ya la conocemos, puesto que estos ejercicios ya fueron resueltos en Racket, pedimos que se implementen una resolución *no recursiva* de los mismos.

Ejercicio 9. Ejercicios transformando una lista como **map**, pero en `Python`.

Estos ejercicios corresponde con los números: 19, 20, 23, 24, 25 de la práctica 5 de `Racket`.

1. Diseñe la función `raices`, que dada una lista de números, devuelve una lista con las raíces cuadradas de sus elementos.
2. Diseñe una función `distancias` que tome una lista de puntos del plano y devuelva una lista con la distancia al origen de cada uno.
3. Diseñe una función `cuadrados` que tome una lista de números y devuelva otra lista donde los elementos que aparezcan sean el cuadrado de los elementos de la lista original.
4. Diseñe una función `longitudes` que tome una lista de cadenas y devuelva una lista de números que corresponda con la longitud de cada cadena de la lista original.
5. Diseñe la función `convertirFC`, que convierte una lista de temperaturas medidas en *Fahrenheit* a una lista de temperaturas medidas en *Celsius*.

Diseñe una función cerca que tome una lista de puntos del plano (representados mediante estructuras posn), y devuelva la lista de aquellos puntos que están a distancia menor a MAX, donde MAX es una constante de su programa.

Ejercicio 10. Ejercicios clasificando una lista como **filter**, pero en `Python`.

Estos ejercicios corresponde con los números: 13, 14, 16y17 de la práctica 5 de `Racket`.

1. Diseñe una función `pares` que tome una lista de números `l` y devuelva una lista con los números pares de `l`.
2. Diseñe una función `cortas` que tome una lista de strings y devuelva una lista con aquellas palabras de longitud menor a 5.
3. Diseñe una función `cerca` que tome una lista de puntos del plano (representados mediante tuplas), y devuelva la lista de aquellos puntos que están a distancia menor a max, donde max es también es un argumento de la función.
4. Diseñe una función `positivos` que tome una lista de números y se quede sólo con aquellos que son mayores a 0.

Ejercicio 11. Ejercicios operando con los elementos de una lista como **foldr**, pero en `Python`.

Estos ejercicios corresponde con los números: 9, 10, 11, 12, 26, 27 de la práctica 5 de `Racket`.

1. Diseñe la función **todos-verdaderos**, que recibe como entrada una lista de valores booleanos y devuelve *True* si todos los elementos de la lista son True.
2. Diseñe la función **uno-verdadero**, que recibe como entrada una lista de valores booleanos y devuelve *True* si al menos uno de los elementos de la lista es *True*.

3. Diseñe la función **cant-elementos** que dada una lista, devuelve la cantidad de elementos que contiene.
4. Diseñe la función **promedio**, que devuelve el promedio de una lista de números.
5. Diseñe una función **producto-lista** que multiplica los elementos de una lista de números. Para la lista vacía, devuelve 1.
6. Diseñe una función **pegar-strings** que dada una lista de strings, devuelve el string que se obtiene de concatenar todos los elementos de la lista.

Referencias

[1] Think Python: How to Think Like a Computer Scientist, Allen B. Downey, 2nd Edition, Version 2.2.18.

[2] Algoritmos y Programación I, Aprendiendo a programar usando Phyton como herramienta, Rosita Wachenchauzer et.al., 2016, (sin publicar).