Actividad 2 - Funciones, `for`s y detalles del `print`

Programación en Python - IAI - ECyT - UNSAM

1er cuatrimestre 2020

El objetivo de esta actividad es usar correctamente las funciones,
introducir los ciclos for y dar detalles de formato del print. Además de introducir algunos algoritmo y algunas particularidades de pyhton.
Envíen los ejercicios resueltos individualmente a: `python@unsam.edu.ar` .
Y complete la autoevaluación de la carpeta.
Los ejercicios marcados con un asterisco (*) son optativos.

Ciclos for sobre listas arbitrarias

Un `for` en `Python` puede moverse sobre diferentes *iterables*. Por ejemlo el siguiente código itera sobre las letras que componen una palabra:

```
for letra in "palabra":
    print (letra)
```

y este otra muestra las letras con su número de orden (decimos que *enumera* las letras):

```
for num, letra in enumerate(palabra):
    print(num, letra)
```

tanto la cadena, como el resultado de enumerate son iterables.

Ejercicio 1. *Escriba un programa que enumere los elementos de la lista* `[Pyhton, C++, C, Pascal]`.

Ejercicio 2. *Dada una frase S y usando* `for` *e* `if` *(pero no métodos propios de* `str` *que resuelvan el problema de una), escriba un programa que*

- *Convierta a mayúscula la primera letra de cada palabra de la frase.*[1]
- *Elimine espacios de la frase.*
- *Elimine espacios, convirtiendo la letra siguiente a mayúscula.*
- *Y por último un programa que invierta el proceso: tome la frase pegoteada recien generada y regenere la frase original con espacios y sin mayúsculas.*

Ejercicio 2*. ¿Cuales serían los dos métodos de `str` que resuelven los primeros dos problemas de una?

Ejercicio 3. Buscar un string dentro de otro. *Escriba un programa que, dados dos strings devuelva (si existe) la posición del primero dentro del segundo.*
Por ejemplo: si `s1 = "Paseo el perro por la vereda"` *y* `s2 = "perro"`*, el resultado será* 9.

Ejercicio 4. Tablas de multiplicar. *Escriba un programa que imprima de forma elegante las tablas de multiplicar del 0 al 9 (sug: imprimir* `'\t'` *como separador). Si puede, evite usar la multiplicación, use solo sumas.*

[1] puede usar `.upper()` y `.lower()` para un caracter.

Ejercicio 5. Menú a la carta. *El menú de un restaurante se compone de una entrada, un plato principal y un postre. Las entradas son cinco para elegir (digamos* `ensalada, camarones, guacamole, lengua o humus`*). Análogamente, hay cinco platos principales y cinco postres (arme sus listas segun gustos personales). Usando simplemente ciclos (y no la librería itertools)*

- *Imprima todas las combinaciones posibles de menues que se pueden armar.*
- *Modifique el programa (sin alterar las listas) para que imprima únicamente las combinaciones que una persona vegetariana comería.*
- *Si alguien come todos los días en ese restaurante y no quiere comer el mismo plato dos veces en una semana: imprima cinco menúes de forma que esta persona pruebe todas las entradas, platos principales y postres a lo largo de una semana. (sugerencia: usar* `pop()`*).*

.

`itertools`. Considere el siguiente ejemplo de la función `combinations` del módulo `itertools`.

```
import itertools
print("Las parejas que se pueden formar son:")
for c in itertools.combinations(['Alicia','Bruno','Carlos', 'Diana'],2):
    print("- %s y %s" %(c[0], c[1]))
```

Ejercicio 6. *Observe que en este caso se generan las parejas sin importar el orden (que no importa el orden significa que 'Alicia y Bruno' es equivalente a 'Bruno y Alicia' y entonces aparece solo una de esas dos formas).*

- *Suponga que vamos a armar parejas para viajar en un automovil, de manera que la primer persona listada maneja y la segunda es copiloto. Genere la lista de todas las* parejas ordenadas *que se puede formar entre estas cuatro personas*[2] *(aca sí imorta el orden).*
- *Genere todas las palabras de 2 letras (con o sin sentido, con o sin letras repetidas) que usen solo las letras 'ABCDE'.*
- *Genere todas las palabras de 5 letras (con o sin sentido, sin letras repetidas) que usen solo las letras 'ABCDE*[3]*.*

Ejercicio 7. *Considere una fila con n personas una al lado de la otra.*

Las personas pueden estar infectadas con un virus, ser inmunes o ser suceptibles de enfermarse. Representaremos esta situación con una lista L de longitud n que en cada posición tiene un 0 (inmune), un 1 (suceptible de enfermarse) o un -1 (tiene el virus). Este virus se propaga inmediatamente a toda persona suceptible de enfermarse que tenga a alguien enfermo a su lado. Las personas inmunes, no se enferman.

Escriba una función llamada `propagar` que reciba un vector con ceros, unos y menos unos y devuelva un vector en el que los menos unos se propagaron a sus vecion con uno.
Por ejemplo, con input
`[ 1, 1, 1, 0,-1, 1, 1, 1, 0, 1,-1, 1, 1]` el output debe ser
`[ 1, 1, 1, 0,-1,-1,-1,-1, 0,-1,-1,-1,-1]` y con input
`[ 1, 1, 1,-1, 1, 1]` el output debe ser
`[-1,-1,-1,-1,-1,-1]`.

Ejercicio 8. *Rehaga el ejercicio de la guía anterior que toma una cadena de caracteres y cambia todas las vocales por - usando como único ciclo el comando* `for c in palabra:`

[2] Sugerencia: investigue la función `product` del módulo anterior
[3] Sugerencia: investigue la función `permutations`

Figura 1: Propagación análoga a la del Ejercicio 7

Método de bisección

El método de bisección es un algoritmo de búsqueda de raíces de una función que trabaja dividiendo el intervalo de búsqueda a la mitad y seleccionando el subintervalo que tiene la raíz. Como veremos más adelante, el algoritmo es muy rápido: mucho más eficiente que una búsqueda secuencial con un paso fijo.

Ejercicio 9. *Considere el siguiente código (ver archivo `02-Ejercicios.py` adjunto)*

```
#Busqueda secuencial

x = 25
epsilon = 0.01
paso = epsilon**2
sol = 0.0
totPasos = 0
while abs(sol**2 -x) >= epsilon and sol<x: #Salgo cuando hallo la solucion con
    poco error o cuando me paso
     sol += paso
     totPasos += 1


if abs(sol**2 -x) < epsilon: #si encontre la solucion
     print("La raiz de  %f es %f"%(x,sol))
     print("La solucion fue encontrada en %d pasos"%totPasos)
else:
     print("Luego de %d pasos no encontre la solucion"%totPasos)
```

¿Cuántos pasos tarda en encontrar una raíz de 25? ¿Cuántos pasos tarda en encontrar una raíz de 30? ¿Cuál es la precisión del método? ¿Si quisiera duplicar la precisión (es decir, dividir por dos el error máximo que puedo cometer) cuántos pasos más me tomaría hallar la raíz de 30?

Ejercicio 10. *Considere el siguiente código (ver archivo `02-Ejercicios.py` adjunto)*

```
#Busqueda por biseccion

x = 25
epsilon = 0.0001
totPasos = 0

cota_inf=0
cota_sup=max(x,1.0)
sol = (cota_inf+cota_sup)/2

while abs(sol**2 -x) >= epsilon: #Salgo cuando hallo la solucion con poco error
     print("busco en [%f, %f], valor medio: %f"%(cota_inf,cota_sup,sol))
     totPasos += 1
     if sol**2 < x:
          cota_inf = sol
     else:
          cota_sup = sol
```

```
    sol = (cota_inf + cota_sup) / 2

print("busco en [%f, %f], valor medio: %f"%(cota_inf,cota_sup,sol))
print("La raiz de  %f es %f"%(x,sol))
print("La solucion fue encontrada en %d pasos"%totPasos)
```

¿Cuántos pasos tarda en encontrar una raíz de 25? ¿Cuántos pasos tarda en encontrar una raíz de 30? ¿Cuál es la precisión del método? ¿Si quisiera duplicar la precisión (es decir, dividir por dos el error máximo que puedo cometer) cuántos pasos más me tomaría hallar la raíz de 30?

Búsqueda binaria

Así como podemos buscar una solución de una función, como en el caso anterior, también podemos encontrar la búsqueda secuencial y la búsqueda binaria (que parte el conjunto de posibles valores al medio en cada paso) en la búsqueda de un valor numérico en un vector. La búsqueda secuencial recorre todo el vector (o la lista), comenzando desde el primer elemento y hasta encontrar (o no) el elemento que se buscaba.

Ejercicio 11. *Escriba una función (documentándola adecuadamente, lea* Cadenas de texto de documentación (pg 24) *y* Estilo de codificación (pg 25) *del Tutorial de Python.) tal que, dado un vector y un elemento determine si el elemento está en el vector o no, y si está que devuelva su posición.*

Sin embargo, si los valores del vector están ordenados, podemos utilizar la idea del Ejercicio 10 para descartar la mitad de los valores en una sola comparación. El algoritmo resultante se llama de *búsqueda binaria* y es mucho más eficiente que la búsqueda secuencial.

Ejercicio 12. *Escriba una función (documentándola adecuadamente, es decir, escribiendo su* `docstring`*) tal que, dado un vector ordenado y un elemento realice una búsqueda binaria para determinar si el elemento está en el vector o no, y si está que devuelva su posición.*

Ejercicio 13. *Compare la cantidad de operaciones que realizan ambas funciones*

ingresando los elementos $128, 37, 176, 167, 286$ y 1 y la siguiente lista:

```
P=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
    101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191,
    193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271]
```

Ejercicio 14. *Considere un mazo de cartas dispuestas en una fila, algunas cara-arriba y otras cara-abajo (al azar, digamos).* Dar un paso en el juego *consiste en tomar cualquier carta cara-arriba, ponerla cara-abajo y dar vuelta la carta a su derecha (independientemente de su estado).*

- *Escriba una función que dada una lista C (que tiene 0 si la carta está boca abajo y 1 si la carta está boca arriba (o* `False` *y* `True`*, como Ud prefira)) avance un paso en este juego.*
- *Escriba una función que dada la lista C, itere la función anterior hasta que se acabe el juego.*

Ejercicio 14* ¿Puede asaegurar que para cualquier C inicial el juego siempre se acaba? ¿Cómo está seguro?

Sugerencia: `random.random()` **genera un número al azar entre 0 y 1 con distribución uniforme. En cambio** `(random.random() <0.5)` **es una afirmación que toma valores booleanos (es** `True` **si el valor aleatorio resulta menor que 0.5 y** `False` **si no). Pueden definir si una carta está con la cara arriba según**

```
import random
carta_cara_arriba=(random.random() < 0.5) #Probabilidad 1/2 boca arriba y 1/2
    boca abajo.
```