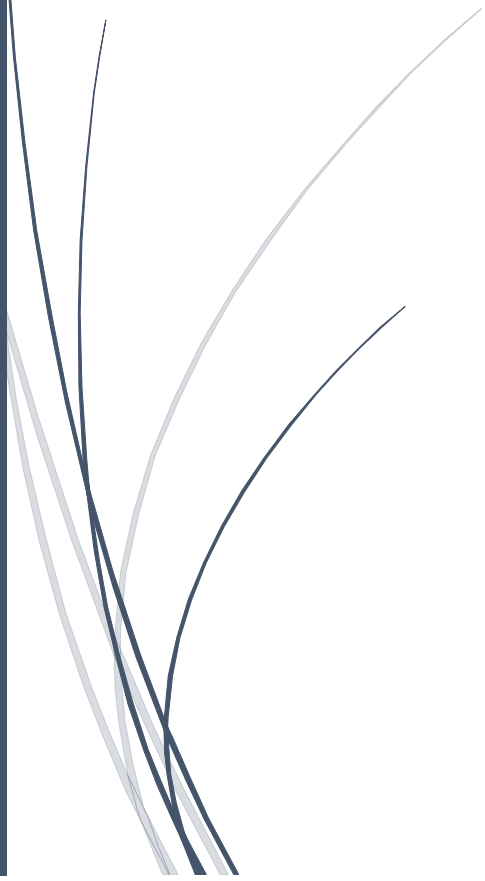
Primera Edición, 2016

Introducción a la programación

Autor:
Juan José Aguillón Ramírez
jjaguillon@uc.cl

Editores:
Prof. Ignacio Casas
Prof. Valeria Herskovic



Prefacio

La programación tiene como propósito crear programas que generen un “comportamiento” para solucionar problemas de la ingeniería.

Ese comportamiento es creado por un algoritmo, o una secuencia ordenada de instrucciones que esta implementado en un lenguaje de programación. Con fines pedagógicos, el curso se dicta en lenguaje de programación Python, el cual tiene por filosofía que sea un código legible y simple con respecto a otros lenguajes, a estos tipos de lenguaje, se les llama “de alto nivel”.

Si bien el libro guía (*) y sus complementos, incluyendo este, se le recomienda al alumno ir a las cátedras y a los laboratorios de práctica que entrega el curso.

El presente compilado, presenta ejercicios resueltos con una explicación para el entendimiento del código que soluciona el problema y el proceso que lleva a esta solución. Cada ejercicio partirá con unas palabras **clave**, lo cual detallará lo que debemos aprender de este ejercicio resuelto.

El compilado viene con las soluciones en formato.py disponibles en el siguiente Dropbox:

https://www.dropbox.com/sh/kle0699vn2tr8h3/AADDsQ1FgkCrkjKRgxEHALFsa?dl=0

Agradecimientos

Este trabajo realizado para la Pontificia Universidad Católica de Chile es un esfuerzo en el cual, directa e indirectamente participaron muchas personas al entregarme su apoyo, opinando, revisando y corrigiendo. Por esas razones, y muchas más, en el presente apartado quisiera demostrarles mi agradecimiento.

En primer lugar, al Ph. D. Jorge Muñoz Gama, quien fue mi profesor en el ramo Introducción a la Programación, el año 2015, mi más amplio agradecimiento por entregarme los conocimientos en esta área que se ha convertido en una nueva pasión a seguir.

Al Ph. D. Ignacio Casas Raposo, quien me otorgó la oportunidad y el apoyo para realizar este trabajo.

A Luis Leiva Sánchez y Eugenio Herrera Berg, compañeros quienes me ofrecieron su apoyo en gran cantidad de este trabajo.

A Allison Bravo, Matías Lillo, Catalina Castro, José Gutiérrez, María José Castro y vuestra perseverancia, que me inspiró para realizar este compilado.

A Daniela Hurtado, quien fue mi tutora, apoyo incondicional todo este tiempo y mi modelo a seguir en el ámbito de ingeniería.

A todos ustedes, mi mayor reconocimiento y gratitud.

Índice General

Introducción al Lenguaje Python

Para introducirnos al lenguaje, debemos conocer y manejar los tipos de datos que puedes ocupar (**string, int, float** y **bool**), el uso de **variables** y manejar las funciones: **input** y **print**.

Problema 1.1:

Pida al usuario el valor de los catetos de un triángulo rectángulo, y entregue el valor de la hipotenusa.

El enunciado pide que el programa interactúe con el usuario, por lo tanto, debemos usar la función input.

```
a = float(input('Ingrese cateto a: '))
b = float(input('Ingrese cateto b: '))
```

Recordar que la función input entrega un dato del tipo **string**. Y para calcular la hipotenusa necesitamos que el dato sea del tipo float, para eso, anteponemos dicho comando que lo transforma.

Luego de tener los datos del tipo float, aplicamos el Teorema de Pitágoras:

$(\text{Cateto\_a})^2 + (\text{cateto\_b})^2 = \text{hipotenusa}^2$

```
c_2 = (a ** 2) + (b ** 2)
c = c_2 ** (1 / 2)
print(c)
```

Problema 1.2:

Pida al usuario un número de cuatro dígitos y calcule la suma de sus dígitos. (Asuma que el usuario solo ingresa números de 4 dígitos)

Para este ejercicio pedimos un numero de 4 dígitos al usuario, como dice el enunciado, podemos asumir que siempre nos entregarán un número de cuatro dígitos, la importancia de esto es que no hay que comprobar dichos casos donde no se cumple esa condición.

```
num = int(input('Ingrese numero: '))
```

Para separar los dígitos de la variable num, ocupamos dos operaciones aritméticas de suma importancia para este curso, el **resto** y la **división en parte entera**.

```
p = num // 1000
num=num % 1000
```

Esto nos entregara en la variable p, el primer dígito de izquierda a derecha, y cambiamos la variable num, como el mismo número sin el primer dígito.

Es decir, si num es 1234, p sería 1 y el nuevo num sería 234.

Repetimos el proceso hasta que nos quede el último número (recordando que s es la inicial de segundo, t de tercero y c de cuarto, para evitar confusiones).

```
s = num // 100
num = num % 100

t = num // 10
num = num % 10

c = num

print(p + s + t + c)
```

Pregunta 1.3

Escriba un programa que reciba 2 strings e imprima los dos strings, separados por una coma, y entregue como una variable booleana (True o False) si los strings que recibe son iguales o no.

Este ejercicio ocuparemos datos del tipo **string**.

```
string_1 = input('Ingrese el primer string: ')
string_2 = input('Ingrese el segundo string: ')
```

Luego, recordar que los strings se pueden concatenar (unir) con la suma, y que la función print puede imprimir en pantalla varios elementos si los separamos con una coma **","**.

Además, podemos comparar strings con la igualdad (**==**)

```
print(string_1 + "," + string_2, string_1 == string_2)
```

ERROR TÍPICO:

```
print(string_1 + "," + string_2 + string_1 == string_2)
```

En este caso se compara la suma de **strings en negrita** con el segundo.

```
print(string_1 + "," + string_2 + (string_1 == string_2))
```

En este caso, sumas un string con un **bool**. Lo cual genera un error.

Control de Flujo: Condicionales

Los condicionales son herramientas del lenguaje que nos permite realizar acciones en caso de cumplir ciertas condiciones, en nuestro caso, datos del tipo **bool**. Para eso, ocuparemos los comandos **if**, **elif** y **else**.

Pregunta 2.1

En un laboratorio del curso QIM100A se necesita analizar unas muestras de pH de diferentes vasos.

Para eso, le piden a un grupo de alumnos del curso IIC1103 que escriban un programa que reciba las mediciones que van realizando, y entregar si la solución es ácida, básica o neutra.

Además, si la solución es básica, debe entregar el valor de pOH de la solución.

Para este ejercicio, ocuparemos elementos básicos de la química para realizar este ejercicio. Además, debemos analizar los **casos condicionales** y que variable **booleana** nos sirve para condicionarla.

Los casos condicionales funcionan con la siguiente oración lógica:

"Si <condición>, entonces <acción>"

Por ejemplo, en este ejercicio: *"Si el pH es menor a 7, entonces la solución es ácida"*

Al ocupar estas sentencias lógicas podemos ordenar mejor los casos condicionales.

```
ph = float(input('Ingrese el valor de la medición: ))

if ph < 7:
   print('La solución es ácida')
```

Caso análogo para pH neutro, es decir, cuando el pH es 7.

```
elif ph == 7:
   print('La solución es neutra')
```

Ahora nos hacemos la siguiente pregunta, *¿cuántos casos condicionales quedan?*

En caso de que solo quede un **único** caso condicional, podemos usar el comando **else**.

Recordar que el valor de pOH es de 14 – pH.

```
else:
   print('La solución es básica, su valor de pOH es:', 14-ph)
```

Ejercicio 2.2

Un corredor se encuentra corriendo la Maratón de Santiago 2016. La maratón tiene 42.195 km (con punto decimal), y en ciertos puntos de la carrera hay carteles indicando los kilómetros recorridos junto a un reloj que indica las horas y minutos transcurridos desde el inicio de la carrera. Nuestro corredor desea terminar la carrera en 4 horas o menos, y cada vez que ve un cartel quiere saber si va bien o si debe correr más rápido.

Escribe un programa para ayudar a nuestro amigo. Tu programa debe recibir un número real positivo menor o igual que 42.195 que indica la distancia recorrida en kilómetros; luego debe recibir dos enteros positivos indicando respectivamente las horas y minutos transcurridos. Con esta información tu programa debe imprimir uno de los siguientes mensajes en pantalla.

- "Ya pasaron las 4 horas. Lo siento."
- "Faltan K kilometros para terminar. Corriendo asi llegaras en H horas y M minutos. Sigue asi."
- "Faltan K kilometros para terminar. Debes ir mas rapido y subir tu velocidad a V km/h para llegar en 4 horas o menos"

Los mensajes cumplen que K y V son números reales positivos, H es un entero mayor o igual a 0, y M es un entero entre 0 y 59, inclusive. No es necesario que tu programa valide que la entrada dada por el usuario es legal. La fórmula para calcular la velocidad es V = distancia/tiempo. Dada una velocidad V y una distancia D, el tiempo para recorrer D a esa velocidad se calcula como D=V. Abajo hay tres ejemplos de ejecución (son independientes; no necesitas poner una iteración en tu programa):

```
>>>
**

MARATON DESANTIAGO
**
Distancia recorrida?: 40.32

Hora?: 5
Minutos?: 23
Ya pasaron 4 horas. Lo siento.
>>>
**
MARATON DE SANTIAGO
**
Distancia recorrida?: 21.0975
Hora?: 2
Minutos?: 0
Faltan 21.0975 kilometros para terminar. Corriendo asi llegaras en 2 horas y 0
minutos. Sigue asi.
```

```
>>>
**
MARATON DE SANTIAGO
**
Distancia recorrida?: 22
Hora?: 2
Minutos?: 50
Faltan 20.195 kilometros para terminar. Debes ir mas rápido y subir tu
velocidad a 17.31km/h para llegar en 4 horas.
```

Fuente: I1 - 2015, 2º semestre.

Para este ejercicio debemos tener en cuenta los **casos lógicos de condición**.

La estrategia que ocupamos es identificar los casos lógicos de condición dados en el enunciado, el resto es solo identificar como calcular lo que te piden.

Identificamos los siguientes del enunciado:

- *"Si la distancia recorrida es menor a 42.195, entonces ejecuta el resto del programa"*
- *"Si la hora es mayor o igual a 4, entonces imprima el mensaje"*
  "Si no, entonces ejecute el resto del programa"
- *"Si a la velocidad que va es óptima para llegar en menos de 4 horas, entonces calcule el tiempo en que llegará e imprima el mensaje"*
  "Si no, entonces calcule a qué velocidad debe ir, e imprima el mensaje."

Con esta estrategia, nos ahorramos problemas de indentación, también se recomienda hacer un diagrama de flujo.

Ahora codificamos en lenguaje Python.

```
print('**\n MARATON DE SANTIAGO\n **')
distancia_recorrida = float(input('Distancia recorrida?: '))
```

El primer caso lógico es comprobar que la distancia recorrida sea menor a 42.195.

```
if distancia_recorrida<=42.195:
  hora = int(input('Hora?: '))
  minutos = int(input('Minutos?: '))
```

Luego, **bajo la misma indentación**, colocamos el siguiente caso lógico.

```
if hora>=4:
    print('Ya pasaron 4 horas. Lo siento')

else:
    distancia_falta = 42.195 - distancia_recorrida
    print('Faltan', distancia_falta,'kilometros para terminar.', end=' ')
```

Para el último caso lógico, debemos calcular lo siguiente:

- el tiempo en horas
- la velocidad (media) actual
- horas que faltan si viajas a esa velocidad
- horas que me quedan antes de cumplir las 4 horas
- Separar las horas que faltan en horas y minutos
- velocidad optima

```
    horas = hora + (minutos / 60)
    velocidad_actual = distancia_recorrida  / hora
    tiempo_falta = distancia_falta / velocidad_actual
    horas_sobran = 4 - horas
    hora_f = int(tiempo_falta)
    min_f = int((tiempo_falta - hora_f) * 60)
    velocidad_optima = distancia_falta / horas_sobran


    if tiempo_falta<=horas_sobran:

        print('Corriendo asi llegaras en', hora_f, 'horas y ', min_f, 'minutos. Sigue asi.')

    else:

        print('Debes ir mas rapido y subir tu velocidad a', velocidad_optima, 'km/h para llegar
en 4 horas o menos')
```

Ejercicio 2.3

Su profesor le envía una tarea, en la cual debe crear un programa que adivine el número en el que esté pensando, y solo le permite realizar sólo 4 preguntas. Tampoco le permite realizar el mismo tipo de pregunta 2 veces.

El profesor le dice una única pista: "Es un número entre el 1 y el 10".

Pero también nos dice lo siguiente: "Escribiré el número en una hoja, su programa debe recibirlo y comprobar que no esté mintiendo en las preguntas"

Para este ejercicio, debemos manejar las variables booleana, para simplificar la búsqueda del número. Puesto que, debemos tratar de eliminar la mayor cantidad de números con cada pregunta y no podremos repetirlas. Las preguntas que nos dividen la cantidad de números a comparar son:

- ¿Es un número par o impar?
- ¿Es mayor o igual a 5?

```
numero = int(input('Ingrese numero (entre 1 y 10) que debo adivinar (no se preocupe, no lo mirare): '))

primera_pregunta = input('Su numero es par o impar?: ')

par = True
if primera_pregunta == 'par':
    par = True
elif primera_pregunta == 'impar':
    par = False

segunda_pregunta = input('Su numero es mayor o igual a 5?: ')

mayor = True
if segunda_pregunta == 'si':
    mayor = True
elif segunda_pregunta == 'no':
    mayor = False
```

Luego, tendremos que decidir nuestra pregunta según las preguntas anteriores.

- Para el caso de que sea impar: Existen 3 números primos entre 1 y 10.
- Para el caso de los pares: Existen 2 números múltiplos de 4

Luego de tener dividido los números, comparamos las respuestas y utilizamos nuestra última pregunta en caso de duda, preguntar directamente si es un número.

```
if not par:
    tercera_pregunta = input('Su numero es primo? (1 no es primo): ')
    if tercera_pregunta == 'si':
        primo = True
    else:
        primo = False

else:
    tercera_pregunta = input('Su numero es multiplo de 4?: ')
    mult_4 = True
    if tercera_pregunta == 'si':
        mult_4 = True
    elif tercera_pregunta == 'no':
        mult_4 = False
```

Luego comparamos los casos:

```
if not par:
    if not mayor:
        if primo:
            res = 3
        else:
            res = 1
    else:
        if not primo:
            res = 9
        else:
            pregunta = input('es 5?: ')
            if pregunta == 'si':
                res = 5
            else:
                res = 7
```

```
else:
    if not mayor:
        if mult_4:
            res = 4
        else:
            res = 2

    else:
        if mult_4:
            res = 8
        else:
            pregunta = input('es 10?: ')
            if pregunta == 'si':
                res = 10
            else:
                res = 6

if res == numero:
    print('El numero en el que usted penso es:', res, 'y el que escribio al principio es', numero,
'¡¡he adivinado!!')

else:
    print('En una de las preguntas me ha mentido, lo se, soy adivino')
```

Control de Flujo: Iteración

Las iteraciones nos sirven cuando un proceso del código se repite hasta que se cumpla una cierta condición, en el caso del **while**, mediante un dato del tipo **bool**. Por otra parte, el **for**, repite el proceso una cantidad n-esima dentro de un rango.

Ejercicio 3.1

Escribe un programa que reciba tres números enteros positivos e imprima todos los múltiplos del número menor que se encuentren entre los otros 2 números recibidos.

Para este tipo de ejercicios debemos identificar el tipo de iteración que nos conviene usar. Para este caso, ocuparemos ambos con fines pedagógicos.

Iniciamos recibiendo los números mediante el **input**.

```
a = int(input('Ingrese el primer numero: '))
b = int(input('Ingrese el segundo numero: '))
c = int(input('Ingrese el tercer numero: '))
```

El ejercicio, además nos pide ordenarlos de menor a mayor.

```
if a <= b and a <= c:
    minimo = a
    if b <= c:
        maximo = c
        medio = b
    else:
        maximo = b
        medio = c
elif b <= a and b <= c:
    minimo = b
    if a <= c:
        maximo = c
        medio = a
    else:
        maximo = a
        medio = c
else:
    minimo = c
    if a <= b:
        maximo = b
        medio = a
    else:
        maximo = a
        medio = b
```

Ahora veremos la solución con el comando **while**.

Primero debemos encontrar el **condicional de iteración**, la condición que se debe cumplir para que se repita un proceso.

Para facilitar el proceso, ocuparemos el adverbio "*Mientras*", de la siguiente forma:

"Mientras <Condición>, hacer <Acción>".

Para este ejercicio, esta es la condición de iteración:

"Mientras el múltiplo del mínimo sea menor o igual al máximo, si el múltiplo es mayor al número medio, imprimirlo en pantalla"

Ahora, codificamos en leguaje Python. Recordar siempre partir con un índice de inicio.

```
i=1
while minimo * i <= maximo:
    if medio <= minimo * i:
        print(minimo * i)
    i += 1
```

Para la otra solución, debemos encontrar un **rango de trabajo**. En este caso, el rango de trabajo está entre el número medio y el máximo. Para eso, ocupamos el comando **range**, el cual debe recibir, el número de inicio y el de termino. Recordar que el número de termino no es incluido en el rango de trabajo, para eso le sumamos 1.

```
for i in range(medio, maximo + 1):
    if i % minimo == 0:
        print(i)
```

Ejercicio 3.2

Los números binarios son aquellos que se representan en base 2. Por ejemplo, el número 29 es posible escribirlo como $2x10^1 + 9x10^0$ en base 10, el cual al representarlo en base 2 sería $1x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$, por lo que el número 29 en representación binaria sería 11101. Como puedes notar, cualquier número en base n se puede escribir sólo con cifras entre 0 y n-1.

Con esta lógica, escriba un programa que reciba un número entero en base 10, y que lo transforme a un número de base ternaria (base 3).

Para este ejercicio, debemos buscar el exponente más alto, por el cual partir dividiendo, en otras palabras, buscaremos la potencia de 3 que más se acerque al número recibido en base 10.

```
num = int(input('Ingrese numero: '))

i=0
while (num // 3 ** i) != 0:
   i += 1
```

Luego, recorremos el rango de trabajo desde el exponente que encontramos, hasta el 0.

Recordar: el for puede **recorrer** el rango **a la inversa** si al final del rango le agregamos una **“,”** y un **“-1”**.

```
num_base = 0
for k in range(i, -1, -1):
   agregar = num // 3 ** k
   num = (num % (3 ** k))
   num_base = (num_base * 10) + agregar
print(num_base)
```

Ejercicio 3.3

Crea un programa que permita dibujar un rombo a partir de su ancho, el cual es ingresado por el usuario. Por simplicidad, debes dibujar un rombo solamente si el número ingresado es impar, e indicar al usuario que no puede dibujar el rombo si el número es par.

Nota: Se puede imprimir varias veces un texto multiplicándolo por un número. Por ejemplo,

print("o"*4) imprime “oooo” y print("hola"*2) imprime “holahola”.

Ejemplos de ejecución (independientes entre sí):

```
>>>
Este programa dibuja un rombo
Ingrese el ancho del rombo (debe ser impar): 7
    *
   ***
  *****
*******
  *****
   ***
    *
>>>
Este programa dibuja un rombo
Ingrese el ancho del rombo (debe ser impar): 6
Ingreso un numero par; no se como dibujarlo!
```

Fuente: I1, 2015 – 2º Semestre

Para este ejercicio trabajaremos bien el **rango de trabajo** del **for**, además, aprovecharemos el reconocimiento de patrones en el ejemplo.

Primero comprobemos que el número recibido no sea par.

```
print('Este programa imprime un rombo')
ancho = int(input('Ingrese el ancho del rombo (debe ser impar): '))

if ancho%2 == 0:
   print('Ingreso un numero par; no se como dibujarlo!')
```

Luego dividiremos el trabajo en dibujar dos triángulos.

Para eso trabajaremos con los string " " (espacio) y el string "*", estos multiplicados por un índice.

Como podemos observar del ejemplo, los espacios están en relación al ancho del rombo, específicamente, la cantidad de espacios es el (ancho//2), y se les resta 1 por cada ciclo.

Luego, siempre parten con un string "*", y se le van sumando 2 por cada ciclo. Ahora, codificamos en lenguaje Python.

```
else:
    for i in range((ancho // 2) + 1):
        print(" " * ((ancho // 2) - i) + '*' * (1 + (2 * i)))
```

Luego, hacemos el trabajo inverso, para dibujar el triángulo invertido.

```
    for j in range(ancho // 2):

        print(" " * (1 + j) + '*' * ((ancho - (2 * (j + 1)))))
```

Funciones

En este capítulo introduciremos la utilización de funciones pre-definidas, ocupando el comando **import**. Dentro de las librerías que ocuparemos en el curso, están la librería **math**, y la librería **random**.

Por otra parte, ocuparemos funciones que nosotros programemos, mediante el comando **def**.

Debemos reconocer que **parámetros** deben ir en la función, es decir, de qué manera reutilizaremos el código dentro de la función.

Ejercicio 4.1

Una ecuación cuadrática tiene la forma de $ax^2 + bx + c = 0$. Escriba un programa que reciba los parámetros *a, b, c* de la forma anteriormente mencionada. Y compruebe si la función tiene solución en los reales, si tiene 2 soluciones, 1 o ninguna.

Recuerde que para comprobar lo del enunciado debe calcular $b^2 - 4ac$ y comprobar si esta expresión tiene como resultado un número, positivo, negativo o neutro.

Además, utilizando la librería *math*, calcule el valor real de *x* dependiendo de cada caso.

<u>Recordar</u>: Para calcular el valor de x se debe utilizar la expresión

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Para este ejercicio, debemos importar la librería **math**.

```
import math

a = float(input('Ingrese constante a: ') )
b = float(input('Ingrese constante b: ') )
c = float(input('Ingrese constante c: ') )
```

Luego con manejo de variables, calculamos el determinante $b^2 - 4ac$.

```
determinante = (b ** 2) - (4 * a * c)
```

Luego, revisamos los casos donde hay más de una solución.

Llamamos una función de la librería **math.sqrt(x)** donde cumple la función matemática *raíz cuadrada*.

```
if determinante>0:
   print('La ecuación tiene 2 soluciones reales')
   x1 = ((-1 * b) + math.sqrt(determinante)) / 2 * a
   x2 = ((-1 * b) - math.sqrt(determinante)) / 2 * a
   print(x1)
   print(x2)
```

Para los otros casos, se procede de la misma manera.

```
elif determinante==0:
   print('La ecuación tiene 1 solución real')
   x1 = ((-1 * b) + math.sqrt(determinante)) / 2 * a
   print(x1)

else:
   print('La ecuación no tiene soluciones reales')
```

La librería **math** también tiene como funciones:

- **log10(x)** *logaritmo de x en base 10*
- **exp(x)** *exponencial de x*
- funciones trigonométricas como **cos(x)**, **tan(x)**, entre otras

Además tienen las constantes **pi** y **e** (número de Euler) disponibles.

Ejercicio 4.2

El "piedra, papel o tijera" es un juego chino que trascendió fronteras por su simpleza y potencial lúdico en el azar.

Sin embargo, la serie "*The big bang theory*", creo un juego inspirado en la simpleza del juego "piedra, papel o tijera", pero agregándole dos agentes "Lizard and Spoke".

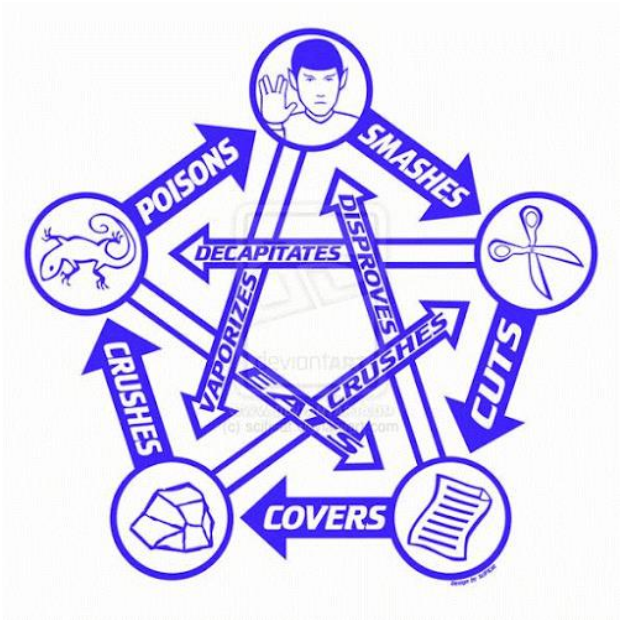


Imagen: Scificat (devianart)

Escriba un programa que reciba uno de estos comandos y que aleatoriamente el computador elija uno de estos, luego, este debe imprimir el ganador.

Además, debes preguntarle al usuario si desea seguir jugando, en caso de una entrada invalida, preguntar otra vez.

Un ejemplo de ejecución:

```
>>>
Bienvenido a Rock, paper, scissors, lizard, Spoke
Que desea jugar?:
[1]Tijeras
[2]Papel
[3]Piedra
[4]Lagartija
[5]Spoke
Jugada: 1

El computador jugo: Spoke

Lo lamento, perdiste

Desea seguir jugando?
[1]Si
[2]No
Respuesta: 2
```

Para este ejercicio debemos pedir una jugada aleatoria, por lo tanto, debemos importar la librería **random**.

Luego, como el juego se repetirá hasta que no queramos jugar, debemos colocarlo dentro de un ciclo, es decir, dentro de un **while**.

```
import random
print('Bienvenido a Rock, paper, scissors, lizard, Spoke')
game_end=False
while not game_end:

    print('''Que desea jugar?:
[1]Tijeras
[2]Papel
[3]Piedra
[4]Lagartija
[5]Spoke''')

    jugada=int(input('Jugada: '))
```

Luego ocupamos la función **randint(a,b)**, que recibe como parametros un rango de numeros enteros, por ejemplo, **random.randint(1,6)** retorna un número entero entre 1 y 6, incluidos estos ultimos.

```
jugada_ia = random.randint(1,5)
  if jugada_ia == 1:
    ia = 'Tijeras'
  elif jugada_ia == 2:
    ia = 'Papel'
  elif jugada_ia == 3:
    ia = 'Piedra'
  elif jugada_ia == 4:
    ia = 'Lagartija'
  else:
    ia = 'Spoke'

  print('El computador jugo:', ia)
```

Una posible solución es hacer **control de flujo**, **condicionando** jugada por jugada.

```
#   if jugada==1:
 #     if jugada_ia==1:
  #        print('Jugaron lo mismo, empate')
   #   elif jugada_ia==2:
    #      print('Felicidades, ganaste!')
    # ...
```

Sin embargo, existe una forma más corta y de mayor simpleza, en honor al nivel de la genialidad de los escritores de la serie *"The big bang theory"*.

El orden y los numeros elegidos para las jugadas no es trivial y nos permitirá, facilmente, clasificar el par de jugadas si es que ganaste, o no.

```
diferencia = jugada_ia - jugada
```

Ocuparemos la diferencia de ambos numeros, pues existe un patrón ante la diferencia y lo mostraremos en las siguiente tabla:

Tu jugada/IA	[1]Tijeras	[2]Papel	[3]Piedra	[4]Lagartija	[5]Spoke
[1]Tijeras	0	1	2	3	4
[2]Papel	-1	0	1	2	3
[3]Piedra	-2	-1	0	1	2
[4]Lagartija	-3	-2	-1	0	1
[5]Spoke	-4	-3	-2	-1	0

En la tabla marcamos con azul los casos donde ganas, con rojo los casos donde pierdes y con negro los casos de empate.

Si los clasificamos por pares e impares, dependerá de si son negativos o no. Sin embargo, solucionamos esto restando 1 a los numeros negativos.

Ocupamos el **% 2** para clasificar paridad.

```
if diferencia < 0:
    diferencia = diferencia - 1

if diferencia == 0:
    print('Jugaron lo mismo, empate')
elif diferencia%2 == 0:
    print('Lo lamento, perdiste')
else:
    print('Felicidades, ganaste!')
```

Finalmente, terminamos preguntando si desea continuar el juego.

```
valida = False
while not valida:
    seguir = int(input('Desea seguir jugando?\n[1]Si\n[2]No\nRespuesta: '))
    if seguir == 1:
        valida = True
    elif seguir == 2:
        game_end = True
        valida = True
```

La librería **random** tiene disponibles funciones que permiten el azar y la simulación en el lenguaje python, entre estas funciones, encontramos:

- **choice(seq)**, recibe como parametro una secuencia de elementos, y elige uno al azar.
- **shuffle(seq)**, recibe como parametro una secuencia de elementos, y entrega la secuencia de manera desordenada.
- **uniform(a,b)**, recibe dos numeros y retorna cualquier numero real entre a y b. Ideal para generar porcentajes de probabilidad en el caso de **uniforn(0,1)**.

Estas funciones las podemos utilizar con datos secuenciales, como listas y strings, que veremos en futuros capitulos.

Ejercicio 4.3

En los supermercados hay una gran cantidad de productos que puedes comprar. Para eso, te piden que programes un consultor de precios, que reciba el nombre y el precio original del producto.

Además, en caso de tener un porcentaje de descuento, que este tambien lo reciba.

Finalmente, este debe entregar un mensaje entregando el nombre del producto y el precio de venta, es decir, el precio con el descuento incluido. Además, este valor debe ser retornado.

Observe los ejemplos:

```
>>>  consultor('Pañales',7999)
Pañales
$ 7999
>>>  consultor('Cloro gel',1490,10)
Cloro gel
$ 1341
```

Para este ejercicio crearemos una función para que podamos reutilizar este codigo con cualquier producto.

Tomando el ejemplo del enunciado, observamos que podemos recibir tanto 2 como 3 parametros, estos parámetros se llaman **parametros por defecto**, los cuales pueden ser omitidos. En este caso, el parametro por defecto es el (**descuento=0)**.

Partimos definiendo la función ocupando el comando **def**.

```
def consultor(nombre, precio, descuento=0):
    precio_venta = int(precio - (precio*descuento/100))
    print(nombre)
    print('$', precio_venta)

    return precio_venta
```

Ejercicio 4.4

Del ejercicio 4.3, el dueño del supermercado te pide que programes una caja para entregar una boleta. Pero como sabemos, una caja es sólo un consultor que imprime una boleta, por lo tanto, aprovecharemos la función "consultor", la cual debemos importar.

La función de la caja es imprimir cada elemento que agreguemos, y finalmente, entregar un total a pagar. Esta función se realizará mientras no agregues un producto "-".

Este ejercicio debemos importar un **modulo** creado por nosotros, en este caso, el archivo se llama modulo.py. el cual tendra la función **consultor**.

Luego, definimos la función **caja()** que al ser llamada, simule los datos que recibe un consultor de precios de una caja.

```
import modulo
def caja( ):
    producto = 'iniciar'
    total_pagar = 0
    while producto != '-':
        producto = input('Producto: ')
        if producto != '-':
            precio = int(input('Precio: '))
            descuento = int(input('Descuento: '))
            total_pagar += modulo.consultor(producto, precio, descuento)
    return total_pagar
```

Strings: cadenas de caracteres

En este capitulo utilizaremos la clase **string**, en conjunto a sus metodos. Además, ocuparemos un nuevo uso para el comando **for**, el cual es recorrer elementos de una secuencia y aprenderemos el uso del operador **in** y la función **len(x)**.

Ejercicio 5.1

Escriba una función que reciba un string de numeros binarios, y retorne la cantidad maxima de 0 y 1 que esten juntos. Un ejemplo de ejecución:

```
>>> cuentabin('1000110')
0s = 3
1s = 2
```

Para este ejercicio, diferenciaremos el uso del **for** sobre strings, cuando usar el for sobre un rango, y cuando usar el for sobre el string.

Para este caso, debemos utilizarlo sobre un rango, puesto que no solo debemos trabajar en un caractér específico, sino que, también sobre los strings que le prosiguen, es decir:

Del string = 'hola mundo'.

- Si queremos encontrar donde esta la primera 'm', trabajamos solo sobre **un caractér**, por lo tanto, ocupamos **for letra in string.**
- Si queremos encontrar la posición donde se encuentra el string 'mun', trabajamos sobre **varios caracteres**, por lo tanto, es recomendable utilizar **for i in range(len(string)),** son el cuidado de no salir del rango del string.

```
def cuentabin(string):
    ceros = 0
    unos = 0
    for i in range(len(string)):
```

Luego colocamos dos contadores y una variable bool auxiliar, para **no salir del rango** del string, además, separamos el termino a comparar, es decír, si contaremos "0's" o "1's".

```
        contador = 0
        head = string[i]
        j = 0
        in_range = True
        while string[i + j] == head and in_range:
```

Luego sumamos cada vez que se cumplan las condiciones del while y comprobamos de no salir del rango.

```
        contador += 1
        if (i+j+1) == len(string):
            in_range = False
        else:
            j += 1
```

Por consiguiente, al salir de este ciclo, clasificamos si contamos “0’s” o “1’s”, además, comprobar si la cadena que acabamos de contar, es mayor a alguna contada anteriormente, si es así, se reemplazará el valor.

Finalmente, imprimir los valores encontrados.

```
    if head == '0' and contador > ceros:
        var = contador
        ceros = var
    elif head == '1' and contador > unos:
        var = contador
        unos = var
print('0s =', ceros)
print('1s =', unos)
```

Ejercicio 5.2

La distancia Levenshtein corresponde al menor número de caracteres que hay que **insertar**, **borrar** o **sustituir** para transformar un string a otro. Por ejemplo:

- Las palabras gato y gatito están a distancia 2, pues para llegar de una a otra basta instertar i e insertar t (o bien eliminar i y eliminar t).
- Las palabras hola y ola están distancia 1; para llegar de una a otra basta instertar h (o borrar h).
- Las palabras gallina y gallina están a distancia 0, pues son iguales.
- Las palabras caro y cara están a distancia 1; para llegar de una a otra basta sustituir o por a.

En esta pregunta debes escribir un programa que pida dos strings al usuario e imprima si ellos están a distancia de Levenshtein 0, mayor que 1, o igual a 1. Si la distancia es igual a uno, se debe indicar la operación (insertar/borrar, o sustituir). A continuación se muestran tres diálogos que ejemplifícan cómo debiese funcionar tu programa:

- > Palabra 1? jaron
  > Palabra 2? jarron
  > Respuesta: 1 operacion (insertar/borrar)
- > Palabra 1? Limon
  > Palabra 2? limon
  > Respuesta: 1 operacion (sustituir)
- > Palabra 1? jarron
  > Palabra 2? melon
  > Respuesta: mas de 1 operacion

Fuente: Midterm – 2014, 2º Semestre

Para este ejercicio, debemos pedir dos strings al usuario, luego, en caso de existir **espacios** de sobra, **eliminarlos** con el método **.strip()**.

```
pal_1 = input('Palabra 1? ').strip()
pal_2 = input('Palabra 2? ').strip()
```

Un caso básico, es cuando las palabras son iguales, es decir, no se aplica ninguna operación al string.

```
if pal_1 == pal_2:
   res = '0 operaciones'
```

Luego, el caso donde se realizan sustituciones, es decir, cuando el largo de la palabra es igual, pero un caractér es diferente. Para eso, ocuparemos el for de rango, puesto que trabajamos sobre más de una letra al mismo tiempo, así, contaremos cuantas letras se reemplazaron.

```
elif len(pal_1) == len(pal_2):
    contador=0
    for i in range(len(pal_1)):
        if pal_1[ i ]  != pal_2[ i ]:
            contador += 1

    if contador == 1:
        res = '1 operacion (sustituir)'
    else:
        res = 'mas de 1 operación'
```

Finalmente, el caso donde se agrega (o elimina un caractér), ocuparemos el recorrido en rango. Entonces, debemos comprobar si al retirarle alguna de las letras de la **palabra mas larga**, pasa a ser igual a la palabra mas corta. Finalmente, imprimir el resultado.

```
else:
    if len(pal_1) < len(pal_2):
        larga = pal_2
        corta = pal_1
    else:
        larga = pal_1
        corta = pal_2
    inside = False
    for i in range(len(larga)):
        dentro = (corta == ( larga[ 0:i ] + larga[ i+1:len(larga) ] ))
        inside = inside or dentro
    if inside:
        res = '1 operacion (insertar/borrar)'
    else:
        res = 'mas de 1 operacion'

print( 'Respuesta:', res )
```

Ejercicio 5.3

Un palíndromo, es una palabra, número o frase que se lee igual adelante que atrás. Si se trata de un numeral,usualmente en notación indoarábiga, se llama capicúa.

Ejemplos: 161, 2992, 3003, 2882.

Se obtiene el capicúa de un número sumando el número con su reverso, hasta obtener su capicúa. Ej: calcular el capicúa del número 57

57+75=132 -> 132+231=363

El capicúa del número 57 es 363. Todos los números tienen su capicúa.

Programe una función que reciba un string numerico, y calcule su capicúa. La función tambien debe comprobar que el string sea numerico.

En este ejercicio, introduciremos los métodos **.isdigit()** y **.isnumeric()**, los cuales retornan True, si y solo si, el string solo tiene digitos(numero int), o, en el caso de isnumeric(), caracteres numericos (admite numeros de clase float).

```
def capicua(string):
    if string.isdigit( ):                          #tambien sirve el .isnumeric()
        while int(string) != int(string[::-1]):
            string = str(int(string) + int(string[::-1]) )
        return string
```

Archivos: Datos en memoria secundaria

En este capitulo aprenderemos el uso de archivos formato **.txt** a través de la función **open()**, además, dependiendo del formato del archivo, ocupar el método **.readline()** el cual lee la primera linea del archivo, y el comando **.readlines()** (notar la **"s"** al final) que retorna

Ejercicio 6.1

Usted visita su antiguo colegio, y se encuentra que los profesores tienen problemas para entregar las notas, a causa de que la lista de alumnos fue escrita de manera continua, con los apellidos separados sólo por un espacio

Uno de sus antiguos profesores, le pregunta si tienes un método para dejar la lista de manera vertical en un nuevo archivo.

Se te ocurre programar una función que reciba el nombre del archivo, y entregue un archivo con los apellidos separados de manera vertical, así el profesor podrá reutilizar el código para otros cursos.

Ejemplo:

Curso.txt

```
Aguilar AlarconB. AlarconF. Astorga Azocar Baeza BarreraB. BarreraD. BarreraK. Bertoglio
Bisamonte Castro DelRio Diaz Dominguez Espinoza GarciaD. GarciaM. Iglesias InfanteA. InfanteS.
Manriquez Monte Moreno Munoz Navarro Pascual PerezD. PerezS. PerezZ.J. PerezZ.P. Sanchez
Sandoval SilvaP. SilvaS. Valenzuela Vivance Zamora Zapata
```

CursoVert.txt

```
Aguilar
AlarconB.
AlarconF.
Astorga
Azocar
Baeza
BarreraB.
…
```

En este ejercicio, ocuparemos el recorrer linea por linea, a través del método **.readlines()**. Debemos abrir el archivo de entrada de información, y el de salida, **recordar que el de salida, debe tener una 'w'** para escribir en este.

```
def ordenarcurso(archivo):
    entrada = open(archivo.strip() + '.txt')
    salida = open(archivo.strip()+'Vert.txt','w')
    for linea in entrada.readlines():
```

Luego, ocupamos el metodo de strings, **.split()**, para recorrer cada palabra de la linea, y luego escribirla en el archivo de salida con el metodo de archivo **.write().**

Finalmente, cerrar todos los archivos.

```
        for palabra in linea.split():
            salida.write(palabra+'\n')
    entrada.close()
    salida.close()
    return
```

Recordar siempre cerrar los archivos.

Ejercicio 6.2

Una empresa te contrata para generar cuentas de emails para cada uno de sus empleados. Te envian el archivo "empleados.txt" con el siguiente formato:

```
Aguilera Francisco Cristóbal Gerencia
Cortez Gonzalo Francisco Gerencia
García Ignacio Felipe Gerencia
Ortiz Lorena Andrea Gerencia
Sarmiento Ignacia Catalina Gerencia
Alcalde André Tomas
Alfaro Martin Ignacio
Campos Francisca Elisa
Rojas Vicente Pedro
Rojas Víctor Pablo
…
```

Te piden que la cuenta tenga el siguiente formato:

Aguilera Francisco **C**ristobal Gerencia → fcaguilera@gerenciaempresa.cl

Alcalde Andre Tomas → atalcalde@empresa.cl

Debe entregar la lista de correos en el archivo "correos.txt".

Para este ejercicio debemos abrir un un archivo de lectura y otro de escritura.

Creamos una variable de comparación ultimo, que nos servirá para comprobar si un correo se repite, y la variable contador para contar las veces que se ha repetido.

Recorremos las lineas y separamos los datos con el método **.split()**

```
entrada = open('empleados.txt')
salida = open('correos.txt', 'w')

ultimo = “”
contador = 0
for empleado in entrada.readlines():
    empleado = empleado.split()
    user = empleado[1][0] + empleado[2][0] + empleado[0]
    user = user.lower()
```

Luego, comprobamos si el correo fue usado anteriormente. Esto se puede hacer gracias a que la lista de empleados está en orden alfabético.

```
    if user == ultimo:
        contador += 1
        correo = user + str(contador)

    else:
        contador = 0
        correo = '' + user
        var = user
        ultimo = var
```

Finalmente, comprobar si es parte de la gerencia, imprimir en el archivo, y cerrar los archivos.

```
    if 'Gerencia' in empleado:
        correo += '@gerenciaempresa.cl'

    else:
        correo += '@empresa.cl'
    print(correo, file=salida)

entrada.close()
salida.close()
```

Listas: Cadenas de caracteres

Ejercicio 7.1

El bifronte es una palabra o frase que permite un sentido leída de izquierda a derecha y otro distinto leída de derecha a izquierda, lo que diferencia este artificio lingüístico del palíndromo, por ejemplo:

- **Amor – Roma**
- **La mina de sal – La sed animal**
- **Logroñes – Señor gol**

Escriba una función que reciba una lista compuesta por strings y que retorne la lista sin:

1. **Elementos repetidos**
2. **bifrontes (no te preocupes por si estos tienen sentido)**

Para este ejercicio, iniciaremos por eliminar elementos repetidos. Para eso, ocuparemos un acumulador vacío para colocar todos los elementos, pero de manera única. Recorreremos la lista por elemento, y agregaremos con el método **.append(elemento)** a la lista vacía.

```
def elimina_bifronte(lista):
    lista_nueva = [ ]
    for elemento in lista:
        elemento = elemento.strip()
        if elemento not in lista_nueva:
            lista_nueva.append(elemento)
```

Luego, creamos una lista con todos los strings de la lista_nueva, pero sin espacios en medio, para luego ocuparla para revisar la existencia de bifrontes.

```
    lista_sin = [ ]
    for palabra in lista_nueva:
        sin_espacios = palabra.replace(" ","")
        lista_sin.append(sin_espacios)
```

Finalmente, tomamos la lista_nueva y a todas las frases dentro de esta, le quitamos los espacios intermedios, lo invertimos, y revisamos si aquel string esta dentro de la lista_sin. En caso de que está presente en la lista_sin, significa que dicha frase es un bifronte, por lo tanto no la agregamos.

```
    lista_final = [ ]
    for frase in lista_nueva:
        sin_espacios = frase.replace(' ','')
        if sin_espacios[ : : -1] not in lista_sin:
            lista_final.append(frase)
    return lista_final
```

Ejercicio 7.2

En Chiaolún, un tradicional pueblo chino, aún se acostumbra a que mujeres sólo bailan con hombres, y viceversa. Este pueblo tiene muchas más mujeres que hombres y por lo tanto se vuelve complejo organizar una fiesta. Esto es porque allí a las mujeres les gusta mucho bailar y se considera culturalmente inaceptable que alguien que quiere bailar se quede sin hacerlo. Se complica aún más la situación porque a los hombres no les gusta bailar demasiado. De hecho, cada hombre tiene un límite de bailes que está dispuesto a bailar en una fiesta.

El objetivo de esta pregunta es que escribas una función para dar un "plan de baile", si éste existe, para una fiesta en Chiaolún dada una descripción de los invitados. Deberás escribir una función plan que reciba dos listas: una con los nombres de las mujeres asistentes a la fiesta y otra con pares de la forma **[n,c]** donde n es el nombre de un hombre y **c** un número entero positivo, que representa la cantidad máxima de bailes que está dispuesto a bailar. Tu función deberá retornar una lista de pares **[h,m]**, donde h es el nombre de un hombre y m es el nombre de una mujer, expresando el hecho de que "h baila con m". La lista debe ser tal que:

- todas las mujeres bailan, y
- ningún hombre baila más de lo que está dispuesto a bailar.

No importa si en el plan retornado algún hombre no baila (a ellos realmente no les llama la atención el baile). La función deberá retornar una lista vacía si es que no existe un plan de baile (es decir, cuando no es posible que todas las mujeres bailen). Por ejemplo, al llamar tu función de esta manera:

plan(['siugmin','daiyu','fenfang'], [['xiaoxun',1], ['shitong',2]])

ésta podría retornar:

[['xiaoxun','siugmin'],['shitong','daiyu'],['shitong','fenfang']] .

Sin embargo el llamado:

plan(['siugmin','daiyu','fenfang','jiaying'], [['xiaoxun',1], ['shitong',2]])

deberá retornar [].

Fuente: I2 – 2015, 2º Semestre

Para este ejercicio, debemos trabajar con los parametros de la función plan, tomando la lista de hombres, y transformarla a una lista parecida a la lista de mujeres, para eso, ocupando el numero **c** como rango. Así, a través de una lista vacía, agregaremos el nombre del hombre **c** veces a la lista.

```
def plan(lista_mujeres, lista_hombres):
    lista_hombres_2 = []
    for hombre in lista_hombres:
        for i in range(hombre[1]):
            lista_hombres_2.append(hombre[0])
```

Luego, creamos la lista vacía plan y comprobamos que se cumplan las condiciones del enunciado, la cual nos pide que todas las mujeres deben bailar, si esta se cumple porcedemos a agregar los nombres a los planes de baile en pares **[h,m]**, explicados en el enunciado.

```
    plan = []

    if len(lista_mujeres) <= len(lista_hombres_2):
        for j in range(len(lista_mujeres)):
            baile = [ lista_hombres_2[ j ], lista_mujeres[ j ] ]
            plan.append(baile)

    return plan
```

En caso de no cumplirse la condicion, retornará la lista vacía, como se propuso en enunciado.

Ejercicio 7.3

Una matriz es el orden rectangular de elementos algebraicos que pueden sumarse y multiplicarse de varias maneras.

En el siguiente ejercicio, deberas programar una función que multiplique dos matrices, con el algoritmo

$$C = AB = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

El cual nos pide multiplicar cada elemento de cada **fila** de la matriz A, con cada elemento de cada **columna** de la matriz B.

Recordar que si la cantidad de elementos de la fila es diferente a la cantidad de elementos en la columna, la matriz no se puede multiplicar. Además que la cada sub-lista dentro de la lista, debe tener el mismo largo.

Primero debemos comprobar que las listas que estamos multiplicando, efectivamente sean marices, para eso, todas sus filas deben tener el mismo largo.

```
def matrix_mult(matrizA, matrizB):
    cuadrada_A = True
    largo_A = len(matrizA[0])
    for fila in matrizA:
        cuadrada_A = cuadrada_A and largo_A == len(fila)

    cuadrada_B = True
    largo_B = len(matrizB[0])
    for fila in matrizB:
        cuadrada_B = cuadrada_B and largo_B == len(fila)
```

Luego, debemos comprobar su multiplicidad, es decir, si el largo de las filas de la matriz A sea igual al largo de las columnas de la matriz B, si es así, comenzamos a crear la matriz C, correspondiente a la matriz resultado.

```
    if len(matrizA[0]) == len(matrizB) and cuadrada_A and cuadrada_B:
        matrizC = [ ]
```

Luego, por cada fila de la matriz A, aparecerá una fila en la matriz C.

```
    for i in range(len(matrizA)):
        nueva_fila = [ ]
```

Después, por cada columna en la matriz B, debe aparecer un término en cada fila, dicho término se obtiene con la suma de los términos de la multiplicación de la fila con la columna correspondiente.

```
        for j in range(len(matrizB[0])):
            suma=0
            for k in range(len(matrizB)):
                suma += matrizA[ i ][ k ] * matrizB[ k ][ j ]
            nueva_fila.append(suma)
        matrizC.append(nueva_fila)
```

Finalmente, retornamos la matriz correspondiente o un mensaje de que no se puede multiplicar dichas matrices.

```
    return matrizC
else:
    return 'No se puede multiplicar'
```

Ejercicio 7.4

Al don Luis le gusta jugar a la lotería, pero para elegir sus números, estos deben cumplir ciertos requisitos:

Siendo una lista ordenada, cada término sucesor debe valer como mínimo el doble de su antecesor. Un ejemplo es [3, 6, 13].

Su tarea es que dados la cantidad de números a elegir (n) y el número máximo posible (m) y entregar la cantidad de combinaciones posibles para un n y m ingresados por el usuario

Nota: no es necesario encontrar las combinaciones sino solo contarlas.

Primero debemos notar que solo hemos de contar las posibles combinaciones que podemos realizar. Para eso definimos 2 funciones, una que encuentre los números que podemos agregar, y otra que cuente las combinaciones.

Para encontrar los números que podemos agregar, necesitamos la siguiente información:

- El último número agregado.
- La cantidad de números que llevamos agregados a nuestra lista de números, es decir, si llevamos los números [1,4] y buscamos el siguiente, estamos en la "altura" 3.
- Los números n y m que entregue el usuario.

Así, esta calculará las opciones que tenemos para agregar.

```
def agregar(ultimo, altura, n, m):
    ultimo_numero = ultimo
    largo = altura
    agregar = [ ]
    for i in range(ultimo_numero * 2, m + 1):
        if i * 2 ** (n - 1 - largo) <= m:
            if i != 0:
                agregar.append( i )
        else:
            break
    return agregar
```

Luego, de utilizamos esta función para contar todas las posibles combinaciones que podemos realizar. Para eso, encontramos las posibilidades con cada número agregado, y luego las agregamos. Nos ahorramos el paso de realizar cada combinación y nos dedicamos sólo a contarlas en esta función.

```
def controlador(n, m):
    ultimo = [0]
    for altura in range(0, n):
        lista = [ ]
        for i in ultimo:
            aux = agregar(i, altura, n, m)
            lista.extend(aux)
        ultimo = lista
    return len(ultimo)
```

Programación con Objetos y Clases

En este capitulo, introduciremos al concepto de clase, el cual, funciona como una plantilla para la creación de objetos, los cuales tienen un cierto estado y comportamiento.

Llamaremos a estos estados: **atributos**, y a su comportamiento: **métodos**.

El fin de programar orientado a objetos, es poder modelar sistemas o ambientes bajo ciertos comportamientos, en otras palabras, ordenar un sistema y hacerlo más legible a la lectura del usuario.

Ejercicio 8.1

Una empresa de videojuegos lo contrata por su habilidad al programar, y le piden que quede a cargo de la creación de los personajes de dicho videojuego.

El juego consiste de varios personajes, los cuales tienen ciertas caracteristicas, como su nombre, su poder de ataque, su resistencia a los golpes, su velocidad y su vida. Su jefe le pide que la vida sea 100 por defecto.

Además, los personajes interactuan entre estos, de manera que pueda atacar a otro personaje, o ver si es más rápido que otro. Le dicen que ocupe la siguiente formula para el daño.

$$Daño = \left\lfloor \frac{ataque}{defensaOponente} \right\rfloor$$

Para simplificar esta creación, decide programar una clase que genere a dichos personajes del videojuego.

Luego su jefe le pide que cree una función que permita guardar y otra para cargar una lista de personajes de un archivo llamado "data.txt".

Para definir la clase Personaje, uno debe revisar muy bien el enunciado para analizar que atributos debemos crear, para eso definimos el **constructor** (__init__).

Se destacó los atributos que pide el enunciado, sin embargo, no en todos los problemas estarán los atributos y métodos de manera explícita.

```
class Personaje:
    def __init__(self, nombre, ataque, resistencia, velocidad):
        self.nombre = nombre
        self.ataque = ataque
        self.resistencia = resistencia
        self.velocidad = velocidad
        self.vida = 100
```

Luego creamos el método **atacar**, el cual nos pide como parámetro otro objeto de clase Personaje.

```
    def atacar(self, oponente):
        oponente.vida -= int(self.ataque/ oponente.resistencia)
```

Seguimos con la función guardar, la cual recibe como parametro una lista de personajes, luego, imprimimos los parametros pedidos por el constructor en el archivo **"data.txt"**.

```
def guardar(lista_personajes = [ ]):
    archivo = open('data.txt', 'w')
    for personaje in lista_personajes:
        print(personaje.nombre, personaje.ataque, personaje.resistencia, personaje.velocidad, file=archivo)
```

Finalmente, la función cargar, debe hacer el proceso inverso.

Para eso, debemos abrir el archivo y por cada linea, crear un objeto de clase Personaje y guardarlo en la lista.

```
def cargar( ):
    archivo = open('data.txt')
    lista_personajes = [ ]
    for personajes in archivo.readlines( )
        p_lista = personajes.split( )
        lista_personajes.append(Personaje(p_lista[0], int(p_lista[1]), int(p_lista[2]), int(p_lista[3])))
    return lista_personajes
```

Ejercicio 8.2

Un taller mecánico trabaja en vehículos, los cuales tienen una marca, un modelo, una patente, y un propietario (que tiene nombre y teléfono). Los vehículos pueden estar en uno de los siguientes estados: en solicitud de presupuesto (**P**), en diagnóstico por un mecánico para elaborar el presupuesto (**D**), siendo arreglado (**A**), en espera de entrega (**E**), o retirado cuando el propietario ya se lo llevó (**R**). Debes implementar lo siguiente:

a) La clase **Vehiculo**, con los atributos **marca**, **modelo**, **patente** y **propietario**.
b) La clase **Taller**, que posee un único atributo correspondiente a una lista de vehículos, y en la que debes implementar los siguientes 3 métodos:
   I) **recibir_vehiculo(self, vehiculo)** : agrega vehiculo al taller, en estado de solicitud de presupuesto.
   II) a**ctualizar_estado(self, patente, nuevo_estado)**: cambia el estado del vehículo de la patente dada al nuevo_estado señalado. Puedes asumir que el vehículo con la patente indicada existe en el taller.
   III) **entrega_vehiculos(self)**: entrega una lista con los nombres y teléfonos de los propietarios de vehículos que están a la espera de ser entregados. Cada elemento de la lista que entrega la función debe ser una lista compuesta de dos strings **(nombre y teléfono)**.

Contempla en tu solución guardar un estado por cada vehículo.

Fuente: I2 – 2015, 2º Semestre.

Para este ejercicio debemos definir la clase Vehiculo, la cual sólo tiene atributos, basta con solo definirlos.

```
class Vehiculo:
  def __init__(self, marca, modelo, patente, propietario=[ '', '' ]):
    self.marca = marca
    self.modelo = modelo
    self.patente = patente
    self.propietario = propietario
    self.estado = ””
```

Luego, definimos la clase taller con su único atributo, la lista de vehiculos, que en caso de no tener, creamos una vacía.

```
class Taller:
    def __init__(self, vehiculos=[ ]):
        self.vehiculos = vehiculos
```

Ejercicio 8.3

Debes completar el programa que se muestra más abajo. Este programa modela la interacción entre los siguientes objetos: dos **cocineros**, un **mesón**, y dos **mozos** del restorán Don Yadran.

Los **cocineros** ponen platos en el **mesón** para que sean retirados por los **mozos**. El mesón tiene un espacio limitado y los **cocineros** no pueden poner más platos cuando el **mesón** está lleno. Don Yadran es muy famoso y se especializa en solo dos tipos de platos: "lomitos" y "completos". El **cocinero1** solo hace "lomitos" y el **cocinero2** sólo hace "completos".

Los **mozos** sacan platos del **mesón** (si es que los hay) y los ponen en una bandeja distribuidora que los llevan a los hambrientos clientes. El **mozo1** sólo saca del **mesón** platos "lomitos" y el **mozo2** sólo saca platos "completos".

El siguiente programa ejecuta muchas iteraciones de la interacción entre **cocineros**, **mesón** y **mozos**. En cada iteración, cada **cocinero** prepara y pone en el **mesón** una cantidad aleatoria de (entre cero y **P**) platos. En el programa, **P** vale **6** ("lomitos") para **cocinero1** y **8** ("completos") para **cocinero2**. Si un cocinero debe poner **x** platos en el **mesón**, pero en éste sólo caben **k**, con **k < x**, sólo pone **k** platos.

En cada iteración, cada **mozo** saca del mesón una cantidad aleatoria de (entre cero y **S**) platos. En el programa, **S** vale **5** ("lomitos") para **mozo1** y **4** ("completos") para **mozo2**. Si un mozo debe sacar **x** platos del **mesón**, pero en éste sólo quedan **k**, con **k < x**, sólo saca **k** platos.

Debes completar este programa escribiendo las clases **Meson**, **Cocinero**, y **Mozo**, asegurando que funcione a cabalidad. No puedes modificar el código presentado. Debes definir las clases, atributos y métodos necesarios. El método **lleno()** retorna True si el **mesón** está lleno y False en caso contrario. El atributo **faltan** indica la cantidad de platos que no estaban disponibles en el **mesón** para ser retirados por un **mozo** (no se acumulan de una iteración a otra). El atributo **tipo** indica si es "lomito" o "completo". El atributo **lom** indica la cantidad de "lomitos" en el **mesón**. El atributo **comp** indica la cantidad de "completos" en el **mesón**.

programa principal

```
meson=Meson(20)                          #crea_un_mesón_de_capacidad_20_platos
cocinero1=Cocinero("lomito",6,meson)          #crea_cocinero_con_máx._6_lomitos
cocinero2=Cocinero("completo",8,meson)      #crea_cocinero_con_máx._8_completos
mozo1=Mozo("lomito",5,meson)              #crea_mozo_que_retira_máx._5_lomitos
mozo2=Mozo("completo",4,meson)          #crea_mozo_que_retira_máx._4_completos
```

```
t=0                 # ejecuta 50 iteraciones

while t < 50:
   cocinero1.agregaPlatos()
   cocinero2.agregaPlatos()
   print(" Mesón lleno : ", meson.lleno())
   mozo1.retiraPlatos()
   print(" Faltan : ", mozo1.faltan," ",mozo1.tipo)
   mozo2.retiraPlatos()
   print(" Faltan : ", mozo2.faltan," ",mozo2.tipo)
   t = t + 1
   print ("t = "+str(t) + " Meson: " + str(meson.lom) + ", " + str(meson.comp))

# fin de la iteración
```

Fuente: Exámen – 2015, 2º Semestre

Para este ejercicio, sólo debemos programar las clases **Mesón, Cocinero y Mózo. Se recomienda leer y analizar completamente el enunciado y las lineas que ejemplifican su ejecución.**

Para la clase Meson debe recibir como parámetro para su constructor el número máximo de platos que puede tener, siendo este, uno de sus atributos en conjunto a los platos (por separado) que tiene disponible para ser retirados. El método lleno, debe comprobar que el meson no se haya llenado, sumando los platos y comprobando que no sea igual a su máxima capacidad.

Además, como dice el enunciado, se utilizarán números aleatorios.

```
import random

class Meson:
   def __init__(self, maximo):
      self.maximo = maximo
      self.lom = 0
      self.comp = 0

   def lleno(self):
      if (self.lom + self.comp) == self.maximo:
         return True
      return False
```

La clase Cocinero, debe recibir como parámetro el tipo de plato que cocina, la cantidad máxima de platos que puede cocinar, y el meson en el cual esta trabajando. El atributo de agregaPlatos, debe generar un número aleatorio de platos que va a cocinar, y agregarlos mientras el meson no se haya llenado.

```
class Cocinero:
    def __init__(self, tipo, maximo, meson):
        self.tipo = tipo
        self.maximo = máximo
        self.meson = meson

    def agregaPlatos(self):
        platos_agregar = random.randint(0, self.maximo)
        for i in range(0, platos_agregar):
            if not self.meson.lleno( ):
                if self.tipo == 'lomito':
                    self.meson.lom += 1
                else:
                    self.meson.comp += 1
            else:
                break
```

Finalmente, la clase Mozo, debe recibir como parametro el tipo y la maxima cantidad de platos que puede llevar y el mesón donde trabajará. El método retiraPlatos debe retirar los platos del tipo que recibe, o aumentar el atributo faltan, el cual indicará cuantos platos podía haber retirado, pero que no habian.

```
class Mozo:
    def __init__(self, tipo, maximo, meson):
        self.tipo = tipo
        self.maximo = máximo
        self.meson = meson
        self.faltan = 0

    def retiraPlatos(self):
        self.faltan = 0
        platos_quitar = random.randint(0, self.maximo)
        for i in range(0, platos_quitar):
            if self.tipo == 'lomito' and self.meson.lom != 0:
                self.meson.lom -= 1
            elif self.tipo == 'completo' and self.meson.comp != 0:
                self.meson.comp -= 1
            else:
                self.faltan += 1
```

Busqueda y Ordenamiento

Ejercicio 9.1

Una empresa de transporte recibe un informe de parte de uno de sus trabajadores, lamentablemente, dicho informe lo realizó un estudiante en práctica y ordenó la lista de camiones por orden de ganancias.

El informe viene en el archivo "informe.txt" de manera que cada linea contiene la patente de un camión y sus ganancias totales del mes. Ejemplo:

```
GA1324 540000
BCSC13 380500
BBSD12 357000
…
```

La universidad se entera de esto, y te envian a ti a solucionarlo. Te piden que ordenes esta lista pero esta vez ordenando las patentes por su antigüedad (Las con 2 letras son de mayor antigüedad que las de 4 letras, ordenelas de manera que "AA0000" sea menor a "ZZZZ99")

Finalmente entregue el archivo "informeR.txt" con el orden pedido.

Introduciremos el termino **key**, que permite generar un ordenamiento por algun elemento de una sublista, o un atributo de una clase. Abrimos los archivos que utilizaremos.

```
def getKey(item):
    return item[0]

entrada = open('informe.txt')
salida = open('informeR.txt', 'w')
```

Luego, con listas vacías, separamos las patentes por tipo, puesto que ordenarlas directamente nos trará problemas por el valor de los caracteres en ASCII.

```
lista_2_4 = [ ]
lista_4_2 = [ ]

for camion in entrada.readlines():
    camion = camion.split()
    letras = 0
    for letra in camion[0]:
        if letra.isalpha():
            letras += 1
    if letras == 2:
        lista_2_4.append(camion)
    else:
        lista_4_2.append(camion)
```

Finalmente utilizamos el método **.sort()**, y utilizamos el parámetro **key=getKey** para ordenar por el primer termino de la lista con los datos del camión y luego imprimimos en el archivo.

```
lista_2_4.sort(key = getKey)
lista_4_2.sort(key = getKey)

lista = lista_2_4 + lista_4_2

for elemento in lista:
    print(elemento[0], elemento[1], file = salida)

entrada.close()
salida.close()
```

Ejercicio 9.2

Un compañero suyo esta participando en una investigación de la facultad de Matemáticas y Estadísticas, y le pide de manera urgente que ordene los resultados en orden descendente (de mayor a menor) diferentes cantidades de numeros.

Sin embargo, le urge tener los resultados de los numeros mayores, más que de los numeros menores, para eso te pide que utilices una estrategia que te permita hacer eso e imprimir en pantalla constantemente dichos resultados

¿Qué estrategia deberias usar para ordenar esta lista? Argumente

Programe una función, puesto que su compañero deberá reutilizarla con nuevas listas.

Deberá usar **Selection sort**, puesto que ordena termino por termino, priorizando la cantidad mayor en este caso.

Utilizamos la función **max** para buscar el termino mayor, y luego imprimimos. Así, si nos enfrentamos a listas de gran tamaño, tendremos rapidamente el termino mayor.

```
def mi_selection_sort(lista):
    for i in range(len(lista)):
        maximo = max(lista[i:])
        print(maximo)
        m = lista.index(maximo, i)
        lista[m], lista[i] = lista[i], lista[m]

    return lista
```

Ejercicio 9.3

Un equipo de basquetbol te contrata para que les ayudes con un pequeño problema.

Cada partido muchos jugadores sufren lesiones y deben sustituirlos, lo bueno es que tienen una gran cantidad de jugadores para realizar la sustitución. Estos jugadores se retiran al final de la banca, para poder descansar y ser atendidos por el equipo médico.

El problema es que el entrenador ha presentado una gran cantidad de quejas por esto, puesto que a él le encanta tener a sus jugadores en orden numerico y empezar a buscarlos desde la mitad de la banca (una banca muy grande), y luego de cada partido, a causa de las lesiones, sus jugadores se desordenan.

Le piden que realice una función que reciba una lista de jugadores representados por su número. Y luego una función que le facilite el trabajo al entrenador, note la forma que le gusta buscar a sus jugadores.

Recursión

Ejercicio 10.1

Su profesor de matemáticas discretas, le enseña un método para encontrar el máximo comun divisor entre dos números. El cual es utilizar el resto de la división de los números que se estan comparando, y realizar el mismo proceso con el numero menor y dicho resto, hasta que estos sean multiplos.

Es decir = Máximo común divisor entre 77 y 56 ≡ Máximo común divisor entre 56 y 21…

Como se pudo apreciar en el ejercicio anterior, se pueden abarcar los problemas de recursion preguntándose:

i) ¿Cuál es el caso mas básico? ¿Qué hacer con él?
ii) ¿Cómo acercarse al caso base desde los demas casos?

La **situación más simple** en este ejercicio sería que uno de los dos numeros fuera divisor del otro:

```
def gcd(a, b):
  if a%b == 0:
    return b
```

Luego, como dice el enunciado, debemos **repetir el proceso**, pero **cambiando los parámetros** por el numero menor y el resto de esta división.

```
  return gcd(b, a%b)
```

¿Qué ocurre si no cambiamos los parámetros?

En el mejor de los casos, sólo tendras mal el resultado. Por otra parte, puedes entrar en el error de recursión infinita. El cual es repetir el proceso infinitamente, arrojando una pantalla completa de errores

```
File "*", line x, in funcion_recursiva
  return funcion_recursiva ( parametros )
RecursionError: maximum recursion depth exceeded in comparison
```

Para evitar este tipo de errores, debemos modelar bien un arbol invertido con el proceso que estamos realizando.

Ejecicio 10.2

Un estudiante de primer año de Agronomía le pide que le ayudes a ordenar su correo electronico, de la manera más rápida, pues tiene una evaluación en pocas horas, y todo el material lo tiene en su correo.

Sin embargo, aquel estudiante es curioso y quiere aprender el sistema que estas ocupando, para así, reutilizarlo en el futuro. Por eso le enseñas el metodo de quick-sort.

Por suerte, los correos electronicos vienen con una fecha de clase Fecha, la cual te permite ordenarla por día y mes, puesto que solo tiene e-mails de este año por ser de primero.

Primero definimos la clase Fecha con un metodo __str__ para poder imprimir las fechas.

```
class Fecha:
    def __init__(self, dia, mes):
        self.dia = dia
        self.mes = mes

    def __str__(self):
        s = str(self.dia) + '/' + str(self.mes)
        return s
```

Luego, una función que nos simplifique la comparación entre fechas.

```
def ordena_fecha_mayor (fecha_1,fecha_2):
    if fecha_1.mes > fecha_2.mes:
        orden = True

    elif fecha_1.mes < fecha_2.mes:
        orden = False

    else:
        if fecha_1.dia >= fecha_2.dia:
            orden = True
        else:
            orden = False

    return orden
```

Finalmente introducimos el metodo de ordenamiento **quick-sort**, el cual ocupa la recursion para ordenar la lista.

Esta función toma un termino y compara el resto con este, si la fecha es mas nueva que la del termino **"pivote"** le guarda en una lista para volver a repetir el proceso en estas listas de menor tamaño.

Para la recursión, debemos preguntarnos:

"¿Cuál es el caso más básico? y ¿cómo podemos llegar a él a partir de los demás casos?"

El caso más básico, escuando la lista que ordenamos sólo tiene un termino.

```
def quick_email (lista):
    antigua = [ ]
    igual = [ ]
    nueva = [ ]

    if len(lista) <= 1:
        return lista
```

Y el resto de los casos, utilizando el pensamiento de "*quick-sort"* es elegir un termino cualquiera y comparar el resto de los terminos con este. Y luego **repetir el proceso** con las listas generadas.

```
    else:
        pivote = lista[0]
        for fecha in lista:
            if ordena_fecha_mayor(fecha,pivote) and ordena_fecha_mayor(pivote,fecha):
                igual.append(fecha)

            elif ordena_fecha_mayor(pivote,fecha):
                antigua.append(fecha)

            else:
                nueva.append(fecha)

        nueva_ordenada = quick_email(nueva)
        antigua_ordenada = quick_email(antigua)

        return nueva_ordenada + igual + antigua_ordenada
```

Ejercicio 10.3

Resolver una *ecuación diofántica* de la forma:

$$a_1 * x_1 + a_2 * x_2 + \cdots + a_n * x_n = C$$

consiste en encontrar valores dentro de un conjunto *D* de números enteros para las variables x_1 , … , x_n, dados los valores de a_1 , … , a_n y el valor de *C.*

Escribe una función para resolver este tipo de ecuaciones, la que debe ser de tipo recursiva (funciones sólo iterativas no serán evaluadas). De haber una solución, tu función deberá imprimir en pantalla una lista con los valores de x_1 hasta x_n . En caso contrario, deberá imprimir "no hay solucion".

Tu función deberá recibir una lista no vacía de números enteros para los coeficientes $a_1, \dots, a_n$, y un número entero para la constante *C.* Además, deberá suponer que los valores de x_i son números enteros entre -10 y 10, es decir, $D = \{-10, -9, \dots \ , \ 9, 10\}$.

Por ejemplo, si quieres resolver la siguiente ecuación diofántica

$$2x_1 + 3x_2 = 5$$

deberás llamar a tu función usando como argumentos la lista [2, 3] y el número 5, y debería imprimir en pantalla, por ejemplo, [-8, 7], ya que $2\ x\ (-8) + 3\ x\ (7) = 5$. Nota que puede haber más de una solución; basta con que el programa imprima una solución válida (si la hay), no todas las soluciones.

Fuente: Exámen – 2015, 2º Semestre

Para este ejercicio introduciremos el concepto de **backtracking**, el cual recorre todas las posibles soluciones de un árbol de posibilidades. Para cada x de la ecuación tenemos 21 posibles valores, los cuales serán evaluados en la función.

Primero definimos la función que resuelve la ecuación dado los parámetros pedidos: lista de valores de a, el valor de C y una lista de valores de x, la cual por defecto, es vacía.

Nuestro caso base, será cuando toda x tenga un valor asignado. Entonces, calculamos la suma y comprobamos si es igual a C.

```
def resolver(listaA, C, listaX):
    if len(listaA) == len(listaX):
        suma = 0
        for i in range(0, len(listaA)):
            suma += listaA[i]*listaX[i]
        if suma == C:
            return True
        return False
```

Luego nuestro caso recursivo, será recorrer todos los valores que podemos evaluar en x, lo cual realizaremos de manera **iterativa, y luego, realizamos el paso recursivo para cada valor.**

```
    else:
        for i in range(-10,10+1):
            listaX.append(i) # Pongo un valor a la 'x'
            if resolver(listaA, C, listaX):
                return True
            listaX.pop()
        return False
```

Finalmente definimos una función que imprima un **único** resultado, como pide el enunciado, en caso de pedir todas las soluciones, la función que cambia es esta y un par de líneas en resolver.

```
def diofantica(listaA, C):
    listaX = [ ]
    if resolver(listaA, C, listaX):
        print(listaX)
    else:
        print("No tiene solucion")
```

Simulación

En este capítulo, introduciremos una técnica de programación que permite **simular** con ciertos porcentajes una situación específica.

Para eso, utilizarémos todo lo aprendido en los capítulos anteriores, en especial, **funciones**, **objetos** e **iteraciones**.

Ejercicio 11.1

El juego Pokemon GO ha tenido una popularidad enorme en este ultimo tiempo, la aplicación se ha descargado casi dos millones de veces en Android y AppStore.

Un compañero y tu quieren ver que equipo pokemón es el mas fuerte entre los suyos, para eso, deciden que una simulación de batalla es lo escencial.

Primero, deben crear la clase Pokemon, con sus valores de vida, ataque, defensa, velocidad, etc. Luego crear los metodos correspondientes para su ataque, y finalmente, que la simulación tome en cuenta las debilidades de ciertos tipos de pokemones ante otros.

Para la simplicidad del ejercicio, ocuparemos solo 3 tipos de Pokemon: Agua, Fuego y Planta, recordando que los tipo Fuego tienen ventaja sobre los de Planta, estos, tienen ventaja sobre los de tipo Agua, y estos últimos sobre los de tipo Fuego.

Ejemplo:

Charmander 77/90 ataca a Squirtle 90/98

No es muy efectivo, Squirtle recibe 8 de daño.

Squirtle 82/98 ataca a Charmander 67/90

Es muy efectivo, Charmander recibe 23 de daño.

Además solo tendran un ataque con un 100% de que el ataque llegue a su contrincante, y para los entendidos en el juego, solo ataques físicos.

Finalmente, el simulador, cuando un pokemon sea derrotado, debe enviar a su reemplazo el siguiente pokemón.

Para este ejercicio debemos programar las funciones y métodos de la clase Pokemon para que la simulación funcione.

Primero creamos la clase Pokemon con los atributos correspondientes al enunciado.

```
class Pokemon:
    def __init__(self, nombre, tipo, atk, dfs, vel, vida):
        self.nombre = nombre
        self.tipo = tipo
        self.atk = atk
        self.dfs = dfs
        self.vel = vel
        self.vida = vida
        self.vida_total = vida
```

Luego definimos el metodo atacar, que modelará un ataque del pokemón, y la función que definirá el bonus de daño por el tipo de pokemón.

```
    def atacar(self,pokemon2):
        base = (1 + self.atk // (self.dfs * 0.5))
        print(self.nombre, self.vida, ' / ', self.vida_total, 'ataca a', end = ' ')
        print(pokemon2.nombre, pokemon2.vida, ' / ', pokemon2.vida_total)
        extra = bonus(self.tipo, pokemon2.tipo)
        if extra == 1.2:
            print('Es muy efectivo,', end = ' ')
        elif extra == 0.8:
            print('No es muy efectivo,', end = ' ')
        daño = int(base * extra)
        print(pokemon2.nombre, 'recibe',daño, 'de daño')
        pokemon2.vida = pokemon2.vida - daño
```

```
def bonus(tipo1, tipo2):
    if tipo1 == 'Agua':
        if tipo2 == 'Fuego':
            return 1.2
        elif tipo2 == 'Planta':
            return 0.8
    elif tipo1 == 'Fuego':
        if tipo2 == 'Planta':
            return 1.2
        elif tipo2 == 'Agua':
            return 0.8
    else:
        if tipo2 == 'Agua':
            return 1.2
        elif tipo2 == 'Fuego':
            return 0.8
    return 1
```

Luego, las funciones que generaran la batalla entre pokemones y la función que generen el combate con el equipo pokemón completo.

```
def batalla_pkmn(pkmn1, pkmn2):
    while pkmn1.vida > 0 and pkmn2.vida > 0:
        if pkmn1.vel >= pkmn2.vel:
            pkmn1.atacar(pkmn2)
            if pkmn2.vida > 0:
                pkmn2.atacar(pkmn1)
        elif pkmn2.vel > pkmn1.vel:
            pkmn2.atacar(pkmn1)
            if pkmn1.vida>0:
                pkmn1.atacar(pkmn2)

def combate(equipo1, equipo2):
    print('La batalla comienza')
    while len(equipo1) != 0 and len(equipo2) != 0:
        batalla_pkmn(equipo1[0],equipo2[0])
        if equipo1[0].vida < 0:
            equipo1.pop(0)
        elif equipo2[0].vida < 0:
            equipo2.pop(0)
```

Finalmente poblamos el sistema con el equipo que estime conveniente

Ejercicio 11.2

La aerolínea más importante de una isla exótica te contrata por tus habilidades de programador.

Esta isla sólo tiene 3 vuelos hacia las islas más cercanas, pero que tienen una manera extraña de definir sus precios.

- El **primer vuelo** va hacia la isla vecina “Centella”, el precio no ha cambiado, y se mantiene fijo sin importar la demanda y oferta, ni menos la inflación.
  Precio = $100000
- El **segundo vuelo** va hacia la isla central, y el precio del vuelo cambia por razones internas de la isla central, que no tiene nada de paradisiaca como la isla donde trabajará
  Precio = $800000 - 900000
- El último vuelo va hacia la isla “Yo lambda”, la cual es famosa por tener una adivina la cual coloca el precio según los astros.
  Precio = $300000 – 1000000

Otro dato extraño, es que en esta isla siempre tienen disponibles la misma cantidad de asientos en cada vuelo: **500 en el primero, 100 en el segundo, y 20 en el tercero**, y nunca se han logrado coordinar para que los vuelos vayan completos y no se sobrevendan.

Te explican que siempre llegan alrededor de **5 veces** la cantidad de personas por vuelo, y que dependerá del precio del vuelo si estas compran o no.

Un precio **mayor a $800000**, tenemos un catastro del **70% de personas compran** el boleto.

Entre $800000 y $300000, el **40%** de las personas compran boleto.

Y un vuelo más **barato**, sólo el **20%** compra el boleto, el resto sólo viene a comprobar que no hayan cambiado el precio del vuelo.

La aerolínea tiene como política **no vender más** de un **110%** de la capacidad total del vuelo, puesto que un **8% de las personas no llega** a tomar el vuelo.

De esas personas que vienen, un **5% se queda sin cupo** para el viaje.

Te contratan para resolver este problema, te piden que simules **un día** cualquiera en la isla, donde sólo salen **3 vuelos de cada uno**.

Para este ejercicio, debemos tener en cuenta los porcentajes, que son la parte mas importante de la simulación, primero importamos el modulo random y definimos las probabilidades por precio.

```
import random

def probabilidad_por_precio(precio):
    if precio > 800000:
        return 0.3
    elif precio > 300000:
        return 0.6
    else:
        return 0.8
```

Luego, obtenemos los datos del enunciado que son importantes como los porcentajes y generamos un numero aleatorio para simular dicha probabilidad.

```
def simulacion(pasajeros, precio):
    posibles = pasajeros * 5
    ventas = 0
    sin_cupo = 0
    prob = probabilidad_por_precio(precio)
    for i in range(posibles):
        if random.uniform(0, 1) > prob and ventas < pasajeros * 1.1:
            ventas += 1
        elif ventas >= pasajeros * 0.05:
            sin_cupo += 1
    print( "pasajeros sin cupo : " + str(sin_cupo))
    entraron = 0
    for i in range(ventas):
        if random.uniform(0, 1) > 0.08:
            entraron += 1
```

Finalmente comprobamos si el vuelo se sobrevendió o si quedaron vacantes, luego poblamos el sistema de simulación.

```
    if entraron > pasajeros:
        print("vuelo sobrevendido")
    elif entraron == pasajeros:
        print("vuelo completo")
    else:
        print("vuelo con vacantes")


for i in range(3):
    print('Vuelo 1:')
    simulacion(500,100000)
    print('Vuelo 2:')
    simulacion(100, random.randint(800000,900000))
    print('Vuelo 3:')
    simulacion(20,random.randint(300000,1000000))
```