Para más información:

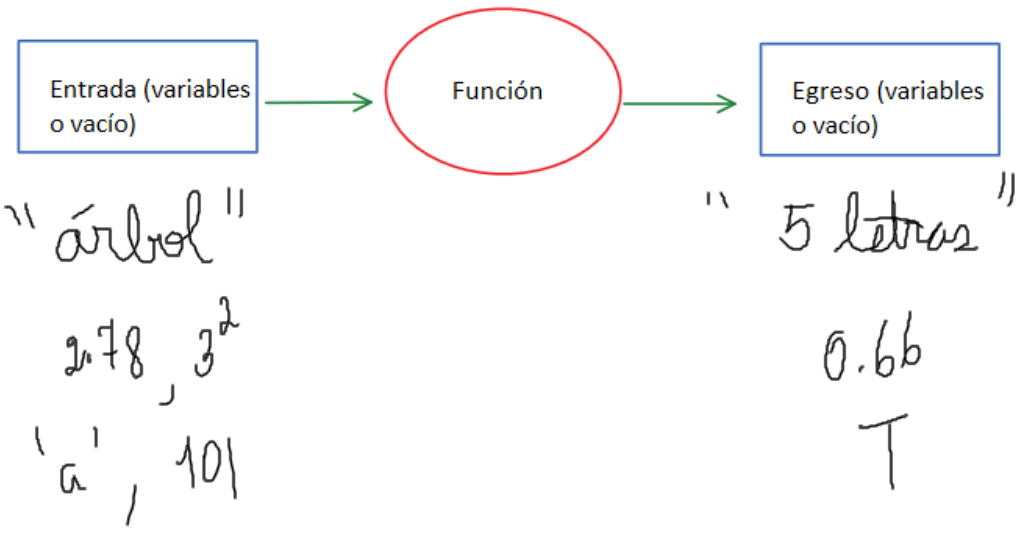
Capítulo 6.2 en delante de Python Programming en http://iran-lms.com/images/images/Books/PDF/Python-Programming_-An-Introduction-to-Computer-Science-Franklin-Beedle--Associates-2016---John-M.-Zelle.pdf

Capítulo 2 de Introduction to Programming Using Java en http://math.hws.edu/javanotes/index.html

Hola a todos y bienvenidos al capítulo 1 en donde exploraremos qué son funciones, para qué sirven y por qué usarlas.

¿Qué es?

¿Entonces qué es una función? Primero vamos a dejar de una lado la definición matemática de una función, la cual establece una relación entre variables. En computación, una función es simplemente un set de instrucciones que nos dice cómo trabajar con un grupo de entradas para devolverme un egreso/salida. En sí, una función es una transformación de entradas que consiste de datos, para que esta me produzca una salida, o egreso. Por ejemplo:

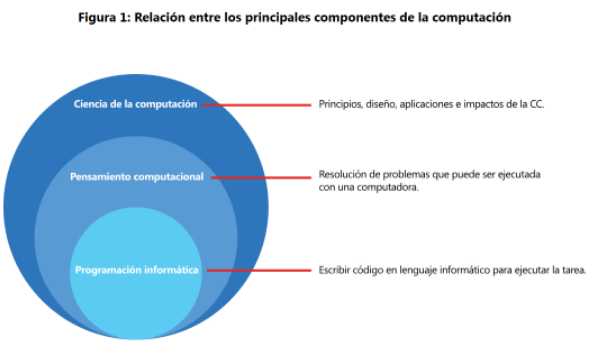


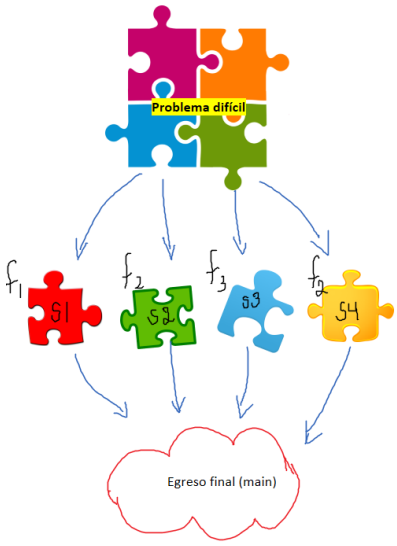
¿Por qué usar funciones?

Ahora podrías estar dudando de por qué usarla. Por ahora nos hemos encontrado con problemas triviales, sin embargo con problemas más grandes y complejos veremos lo importante que es organizar nuestro código en funciones. Además:

1. Definir funciones es una parte integral de programación. Saber cuándo y para qué definir funciones nos permite desarrollar el pensamiento computacional. Por ejemplo, dado un problema complejo, hay que saber cómo descomponerlo en sub-problemas y resolver cada una a través de una función. Esto es para combinar los resultados en una función final

con el fin de resolver el problema complejo. Lo vamos a observar con Java.





Fuente: (Jara & Hepp, 2016)

2. Una función permite generalizar un proceso. Un set de instrucciones puede volverse repetitivo, y definir funciones permite evitar esta redundancia.

3. Esto y muchas más. Permite escribir un código más claro y conciso. Organizar de mejor manera mis ideas, entre otros..

En Java, una función es definida por medio de la siguiente sintaxis:

$$constructor \quad modificador \quad tipo\_var\_egresado \quad nombre\_funcion(tipo\_var \ ingresos)\{$$
$$líneas\ de\ código;$$
$$\text{return } egreso\ \}$$

```
public static double miFuncion(tipo_var nombre_var, ... ) {
        líneas de código

        return egreso;
    }
```

Tener o no el modificador static puede resultar en diferencias muy sutiles. Si tienes la

curiosidad de explorar cuál es la diferencia entre funciones estáticas y no-estáticas puedes visitar: https://www.geeksforgeeks.org/difference-between-static-and-non-static-method-in-java/

En Python el sintaxis es, como podrás haber adivinado, mucho más fácil.

$$\text{def } nombre\_fun\ (ingresos):$$
$$líneas\ de\ codigo$$
$$\text{return } egreso$$

Función en Java:

Problema 1: Interfaz para compra de objeto con descuento

A continuación vamos a resolver el siguiente problema trivial. Supongamos que estás escribiendo un programa para un pequeño robot que le da las bienvenidas a un cliente cuando este visita una tienda. Dado el nombre del cliente y el dinero en su billetera, el robot específicamente dice: "¡Bienvenido a nuestra tienda, *nombre*, por ahora tienes en tu cuenta digital *cantidad de dinero* bitcoins.

Después, cuando el cliente esté decidiendo sí o no comprar un objeto, calcular lo que le sobraría. Este objeto puede tener un descuento. Específicamente, dado el precio del objeto, el descuento dado como entero y el ahorro en la cuenta, el robot dirá: "Si compras el objeto, te quedará *valor que sobra en cuenta*".

(Pista: $sobra = dinero - \left(1 - \frac{descuento}{100}\right) * precio$)

Aquí tendrías 2 opciones:

Killari (luz de luna) es alguien quien tiene 20.4 de ahorro y está decidiendo sí o no comprar un adorno que cuesta 5.5 y tiene un descuento de 60%.

```
print("Bienvenido a nuestra tienda, Killari. Por ahora tienes
en tu cuenta digital 20.4")
sobras = 20.4 - (1 - 60/100) * 5.5
print("Te queda: " + sobras) # 18.2
```

¿Qué pasa si viene Sami? Sami es alguien quien tiene 558.90 de ahorro y está decidiendo sí o no comprar una laptop que cuesta 279.15 que tiene un descuento del 20%.

```
print("Bienvenido a nuestra tienda, Sami. Por ahora tienes en
tu cuenta digital 558.90")
sobras = 558.90 - (1 - 20/100) * 279.19
print("Te queda: " + sobras) #335.55
```

INEFICIENTE.

Usar funciones para simplificar. Observemos en Java.

```
public static double billetera(double dinero, int descuentoEnt
, double precio ) {
        double descuentoDec = 1 - descuentoEnt / 100.0; // div
ision por decilmal

        double pagas = precio * descuentoDec;

        double sobra = dinero - pagas;

        return sobra;

    }
    public static void intro(String nombre, double dinero){

        String intro = "Bienvenido a nuestra tienda, " + nombr
e +
                        ". Por ahora tienes en tu cuenta" + di
nero;

        System.out.println(intro);
    }

    public static void main(String[] args) {

        // Introducción
        intro("Supercalifragilisticespiailidoucious", 80.6);

        // Sobra después de haber comprado
        double result = billetera(80.6, 50, 15.6) ;
        System.out.print("Si compra el objeto le queda " + res
ult);
    }
```

Mencionar qué ocurre cuando borro static en la función billetera.

Problema 2: Chequeo de igualdad

Supongamos ahora que queremos chequear si el cuadrado de la suma es igual a la suma de los cuadrados con respecto a un par de números *a* y *b*. (Pista: en computación, para denotar igualdad en una expresión booleana se usa $>, <, ==$ para denotar mayor que, menor que e igual que respectivamente. Por ejemplo, $a = 18, b = 19; a = 2, b = 2, a = 3, b = 0$.

Podemos usar una función para generalizar en vez de probar uno por uno:

```
    print(18**2 + 19**2 == (18 + 19)**2)
    print(18**2 + 19**2 == (18 + 19)**2)
    print(18**2 + 19**2 == (18 + 19)**2)
```

versus lo siguiente:

```
def chequeo(a, b):
    suma_de_cuadrado = a ** 2 + b**2 ;
    cuadrado_de_suma = (a + b) ** 2;
    miPregunta = suma_de_cuadrado == cuadrado_de_suma;

    return miPregunta
```

```
miFuncion(18, 19)
miFuncion (2, 2)
miFuncion (3, 0)
```

Ejercicios

1. **Define una función que chequea si un entero es divisible para 2 (usa módulo)**
2. **Define una función que dado un decimal que equivale a la temperatura en Farenheit, esta lo convierte en Celsius.**
3. **Define una función que en primer lugar imprime una introducción al programa (tú decides qué mensaje imprimir). Este programa calcula el promedio de 5 decimales.**
4. **Define una función que toma una cadena como entrada e imprime los versos de la canción de cumpleaños con el nombre indicado por la cadena.**
5. **Define una función que toma un entero *n* y obtiene el término en la posición *n* de una sucesión Fibonacci.**
6. **Define una función que dado un entero no negativo *m* imprime la siguiente estructura piramidal:**

1

22

333

4444

….

mmmmm..mm

7. **Dado una lista de números a = [0, 1, 2, … 10] y b = [0, 1, 2… 10], define una función que me dice qué pares de números (a, b) respetan la siguiente igualdad a = 2b + 1**

En Resumen:

Una función es parte integral del arte programar: saber programar es saber definir funciones

Una función generaliza procesos y simplifica código

Varias funciones ayudan a resolver subproblemas de un problema general.

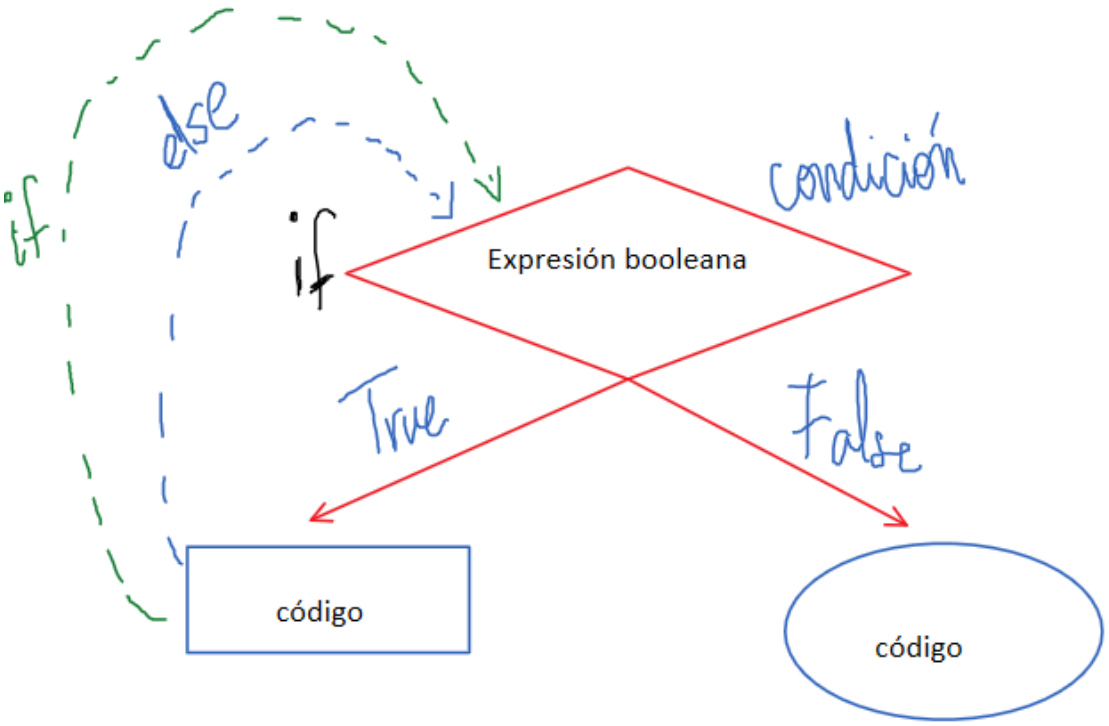
Capítulo 1.1: bucles y estructuras de control

¡Saludos!

En el capítulo anterior concluimos con un pequeño grupo de problemas que no son posibles resolver con lo que sabemos por ahora. Hoy vamos a explorar diferentes estructuras algorítmicas/funcionales con el fin de equiparnos con más habilidades computacionales para resolver problemas más diversos. Estas reciben el nombre de estructuras de control y bucles.

Estructuras de control

Las estructuras de control en programación son para simular **decisión: en otras palabras,** decidir qué líneas de códigos son ejecutados **dado una condición** que hay que satisfacer. Observemos el siguiente diagrama.



En Python, La sintaxis es la siguiente, y el código es ejecutado desde arriba hacia abajo. Si una expresión booleana es evaluada a falso, la siguiente expresión (si la hay es chequeada):

if $expresión\_booleana$: # $primera\ condición$

$código$

elif *expresión_booleana*: # *segunda condición (si la hay)*

código

else: # *demás condiciones (si la hay)*

código

En Python usemos el siguiente ejemplo: chequear si un número es impar. Usemos el operador módulo para chequear si un número es impar o no (puedes dejar preguntas en el foro si no sabes qué es módulo). Siguiendo el diagrama, podemos agregar una estructura de control anidado: chequear si un número es dos e imprimir " es dos, un número primo". Finalmente, de lo contrario es un decimal. El orden va de arriba hacia abajo, y cuando una condición es cumplida, seguirá ejecutándose ar lo menos que se encuentre con un return o break.

```
def es_impar(n):
    if n % 2 == 1:          # el simbolo % representa operacio
n modulo
                            # primera condición
        print("es impar")
    elif n % 2 == 0:        # segunda condición
        print ("es par")
        if n == 2:              # condición anidada
            print("es dos, un primo")
    else:                   # tercera condición
        print("es decimal")
```

El equivalente en Java es el siguiente. Vale la pena mencionar que debido a que el sintaxis de Java es más estricto, estrictamente debe haber una sentencia "devolver (return)" fuera de una estructura de control para una función que no es vacía.

```
public static String es_impar(float n){
        if (n % 2 == 1) {
            return "Es impar";
        }
        else if (n % 2 == 0) {
            System.out.println("Es par");

            if (n == 2) {
                return "Es dos, un primo";
            }
        }
```

```
        else {
            return "Es decimal";
        }
        return null;
    }
```

Bucles: Iteración

Los bucles son usados para ejecutar un mismo bloque de código, un fenómeno llamado iteración. Esto se repite hasta que una condición sea satisfecha. Nosotros vamos a estudiar dos clases de bucles: bucles mientras (*WHILE-LOOP*) y bucles por (*FOR-LOOP)*

Bucles mientras

Los bucles mientras son usados usualmente cuando **no sabemos** precisamente cuántas veces repetimos un bloque de código. La esquematización es el siguiente:

Expresión booleana

código

código

El código instruye cómo manipulas datos hasta que en algún punto la expresión booleana es evaluada a falso. La sintaxis es:

while $expresión\_boolena$:

$código$

```
def ej_mientras():
    i = 0;
    while i < 10:
        print("Python")
        i = i + 1
```

Este es un problema trivial, pero a medida que programamos veremos la funcionalidad de bucles mientras. El equivalente en Java es:

```
int contador = 0;
        while (contador < 10) {
            System.out.println("Java es lo mejor");
            contador = contador + 1;
        }
```

¿Qué pasaría si no escribiésemos la línea `contador = contador + 1;`**?**

Bucles por

Los bucles por son usados usualmente cuando sí **sabemos** precisamente cuántas veces repetimos un bloque de código. La esquematización es el siguiente:

Condición

código

código

Tal como con estructuras de control, los bucles pueden ir dentro de un bucle externo, recibiendo el nombre de bucle anidado.

En Python, un bucle tiene el sintaxis:

$$\text{for } nom\_var \text{ in range}(inicio, final, paso):$$
$$código$$

El código será ejecutado por $\frac{final-inicio}{paso}$ veces.

```
def ej_por():
    for i in range(0, 10, 1): # lo mismo que range(10)
        print(i ** 2)
```

De manera predeterminada, en Python si solo especificas el número máximo de iteraciones, entonces este supone que empiezas a partir de cero e incrementas de uno en uno. Para entender qué está ocurriendo en la expresión por, lo que está pasando es que yo quiero que la variable *i* tome por cada iteración cada uno de los valores de esta $list(range(0, 10, 1))$.

Es importante mencionar que la mayoría de lenguajes de programación realizan un conteo a partir de cero (en vez de 1), el cual es un aspecto importante del pensamiento computacional.

Dado que en realidad una variable está tomando los valores dentro de una lista, un bucle no solo funciona con listas definidas por range. Podemos poner otra lista, por ejemplo:

```
def ej_por_mejorado():
    miLista = [5, 1, 9, 10, -1]
    for num in miLista:
        print(num ** 2)
```

Aquí, la variable num tomará cada uno de los valores de miLista. La longitud de esta lista es 5, por ende en este bucle, la línea de código que imprime los valores será ejecutada 5 veces.

El equivalente de un bucle en Java es:

$$\text{for } (inicialización; expresión\ booleana; paso)\{$$
$$\}$$

```
public static void main(String[] args) {

        int i;
        for (i = 0; i < 10; i ++) {

            System.out.println(i * i);
        }
```

Al igual que en Python, yo también puedo hacer que un bucle-por itere por una lista que yo desee. Estos bucles reciben el nombre de bucle mejorado, y la sintaxis es la siguiente:

```
public static void main(String[] args) {

        String[] list_nombres = new String[] {"C", "C++", "Pyt
hon", "Java"};
        for (String nombre : list_nombres ) {
            System.out.println(nombre);

        } }
```

Tanto con un bucle, o bucle mejorado, en Java será posible imprimir los elementos de una lista, el cual es un problema que te ofrecí en el prólogo y evitar [Ljava.lang.String;@2ff4acd0

En resumen:

Hoy vimos dos estructuras funcionales: estructuras de control y bucles, las cuales son fundamentales para simular decisiones e iteraciones, respectivamente, en programación.

Ejercicios (en cualquier lenguaje de programación, a lo menos que sea especificado)
Algunos ejercicios son traducidos desde: http://www.beginwithjava.com/java/loops/questions.html

1. **Dado un entero m, chequea si este contiene dos dígitos, devolviendo "True". Si tiene más dígitos imprime "tiene más de dos dígitos".**
2. **En java, imprime una lista de números que tú inicialices. Puedes usar un bucle**

mejorado. Haz lo mismo en Python, pero si un bucle.

3. **Dado una lista de enteros (e.g. [3,7,5, 8, 10, 1, 4, 2], imprime con una condición cada uno de los valores de la lista elevado al cuadrado. La condición es que los valores impresos no pueden exceder 25.**
4. **Dado una lista de caracteres (e.g. [‘a’, ‘c’, ‘D’, ‘P’, ‘j’]), solo imprime los valores que están en mayúscula.**
5. **Dado un entero n, imprime una lista de números elevados al cubo, pero de orden descendiente a partir de n.**
6. **Dado un entero n, imprime un sucesión Fibonacci hasta el término *n*.**
7. **Dado un entero n, calcular la suma de la sucesión aritmética 1 + 1/2 + 1/3 + 1/4 + 1/5 +……1/n**
8. **Dado un arreglo de enteros, calcula el promedio de los valores dado en este. Trata de no usar funciones externas como sum() o len().**
9. **Dado un entero m, imprime la siguiente estructura. Lo que está entre paréntesis no es necesario imprimir.**

1111…1 (m veces)

222..2(m-1 veces)

33..3 (m-2 veces)

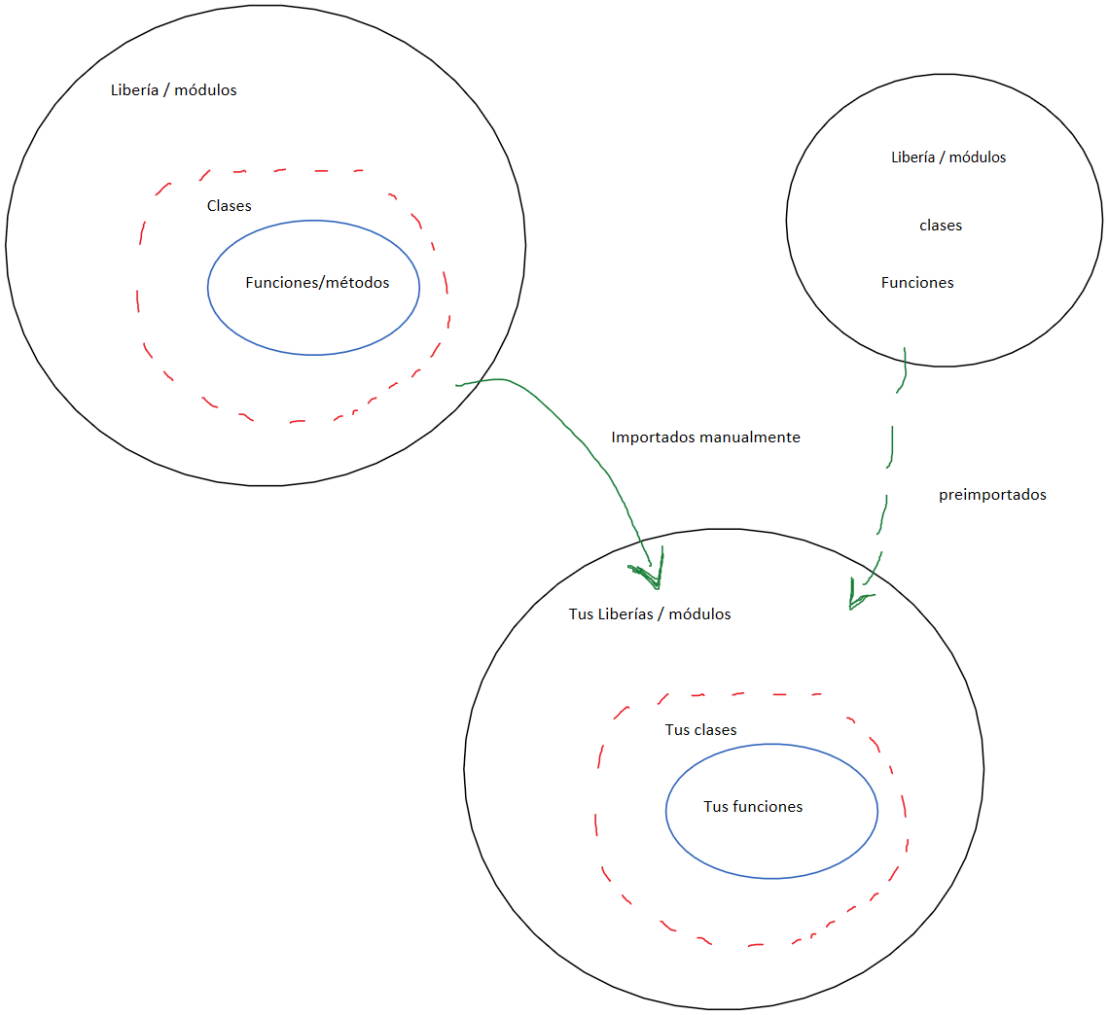
….

m(1 vez)

Capítulo 1.2: funciones predefinidas y librerías externas

Una idea importante a tener en mente es que el arte de programar no es el producto del trabajo individual, sino de una grande, activa y dinámica comunidad. En realidad, casi todos los operadores que hemos utilizado, han sido predefinidos por el grupo de programadores quienes crearon estos lenguajes y facilitado la manera en que escribimos código. Los lenguajes de programación también son constantemente actualizados por programadores quienes están entusiasmados para mejorar y expandir las funcionalidades del lenguaje.

Observemos el siguiente diagrama para ilustrar esta idea:



Hasta este punto, hemos estados definiendo funciones para resolver distintas tareas/problemas. Recordemos que una de las razones por las cuales las funciones son definidas es para evitar repetir líneas de código y generalizar procesos. Nosotros podemos reciclar o reusar las funciones escritas por otras personas que vienen tanto preinstaladas o tienes que instalar a tu lenguaje de programación. También cabe mencionar que en distintos lenguajes de programación, *función* también puede ser llamada "método". Lo mismo ocurre con paquete y librería; creo que se refieren a lo mismo. Puedes revisar https://www.geeksforgeeks.org/what-is-the-difference-between-pythons-module-package-and-library/ si tienes la curiosidad.

Funciones que realizan tareas similares *pueden* estar agrupadas dentro de una misma clase. Y un grupo de clases similares pueden estar agrupadas dentro de una misma librería. Nosotros hemos escrito funciones dentro de un archivo. Un archivo es una clase. Y un grupo de clases están dentro de un directorio, que es como una librería.

Cada vez que creamos un archivo, tanto Python como Java ya importan algunas librerías de manera predeterminada para simplificar su sintaxis de escribir código.

Funciones predefinidas: a trabajar con arreglos y listas

Si deseamos usar una de estas funciones, nosotros podemos **importarlas** a nuestro archivo y usarlas. Algunas funciones y clases ya son importadas de manera predeterminada cada vez

que abrimos un archivo y deseamos programar. Por ejemplo, la función *print* no requiere que hagas una importación. Lo mismo ocurre con algunas variables referenciales, como la lista en Python y los arreglos en Java. Observemos primero en Python.

Algunas de las funciones que ya vienen predeterminadas son las que puedes usar en listas.

```
miLista = [8.9, 1.3, 5.5, -2.4, 3.1, 9.8]
len(miLista) # devuelve la cantidad de elementos en la lista
sum(miLista) # devuelve la suma de los numeros en la lista
miLista[5] # devuelve el elemento en la posicion indicado por el indice
miLista.append(10) # agrega un nuevo numero a la lista
miLista.pop() # remueve el primer elemento, y decrece el tamaño de la lista
miLista.index(8.9) # devuelve el indice de un numero
```

En adición, recordemos que una cadena es simplemente una lista de caracteres, entonces algunas funciones que se usan en listas también funcionan con cadenas.

```
listCaracteres = "Vamos a programar" #lista de caracteres
listCaracteres[8] # lo mismo que el anterior
len(listCaracteres)
listCaracteres.index("a") # devuelve el indice del caracter indicado
listCaracteres.upper() # convierte en mayuscula todos los caracteres
listCaracteres.lower() # convierte en minuscula
```

En Java las listas no son importadas de manera predeterminada, sino que tienes que importarlas manualmente. Lo que sí son importadas de manera predeterminada son los arreglos (inglés: *arrays*). Estás sí tienen algunas funciones útiles que vienen importadas con ellas, sin embargo, no es tan diverso como en Python. Las cadenas también son importadas de manera predeterminada, sin embargo, aquí, en vez de llamarlas lista de caracteres, tendremos que llamarlas “arreglos” de caracteres para ser más preciso.

```
int[] miArreglo = new int[] {12, -9, 5, 0, 8, 11, 10};
miArreglo.clone();
miArreglo.length;


String oracion = "Vamos a programar";
oracion.length(); // devuelve el tamano
oracion.charAt('p'); // devuelve el indice del caracter
oracion.toLowerCase();// convierte todo en minuscula
oracion.toUpperCase();// convierte todo en mayuscula
//oracion[9];

char[] miListaCaracteres = oracion.toCharArray();//lo conviert
e a arreglo de caracteres
```

```
miListaCaracteres[9];
```

En ambos casos no necesitamos memorizarnos qué funciones están predefinidas e importadas. Una lista de funciones predefinidas es proveída por nuestro EDI. Puedes presionar mantener presionado CTRL y cliquear con el mouse para observar el código original.

Importación de librerías externas

Si las funciones predefinidas no son suficientes o simplemente queremos explorar más funciones dependiendo del problema a resolver, siempre podemos importarlas desde librerías externas.

Problema 1: Dado un entero no negativo *m*, genera una lista que contiene *m* números aleatorios y calcula su suma. El rango de números aleatorios es entre 1 y 10.

Hay 3 maneras de importar librerías o funciones externas. Estas 3 maneras difieren en simplicidad. A veces es tedioso llamar una función que contiene nombres o direcciones largos.

```
import random as rd
from random import randint
from random import randint as ri
```

```
def sumListAleatoria(m):
    random.seed(12)
    listaAleatoria = []
    for i in range(m):
        listaAleatoria.append(random.randint(1, 10))
    print(listaAleatoria);
    return sum(listaAleatoria)
```

En java es un poco más complejo, pero la idea de importar una librería externa se mantiene:

```
import java.util.Random;
```

```
public static int listAleatoria(){
        Random rand = new Random(); // crear una instante de l
a clase
        rand.setSeed(15);
        int[] miListaAleatoria = new int [18];
        for (int i = 0; i < miListaAleatoria.length; i++){
```

```
            miListaAleatoria[i] = rand.nextInt(10);

        }
        int sum = 0;
        for (int j : miListaAleatoria) {
            sum += j;
        }
        return sum;
    }
```

Para finalizar, hay una multitud de funciones que podemos importar para distintas razones. Para algunas hay que instalarlas. En el terminal de comando tienes que usar conda install.

```
import math
import numpy
import pandas
import collections
import network
import scipy
import matplotlib
import torch
```

Así como tú puedes crear tus propias funciones dentro de clases y agruparlas en librerías y compartirlas con la comunidad de programación.

En resumen

Podemos importar y utilizar librerías que contienen funciones que nos pueden ser útiles para resolver problemas sin tener que redefinirlas por nuestras cuenta

Puedes tratar de definir tus propias funciones como forma de practicar.

Tú también puedes crear tus propias funciones dentro de una libraría y compartirla con el resto de la comunidad de programadores (ejemplo: GitHub)

Ejercicios

1. Chequea si una lista es palíndroma, lo cual significa que la lista es igual tanto si lo vez de izquierda a derecha o viceversa.

2. Calcula el volumen de una esfera dado el radio. La fórmula a usar es $V = \frac{4}{3}\pi r^3$. Utiliza la librería Math.

Capítulo 1.3: Introducción a depuración

Textos a consultar:

Capítulo 3, sección 3.7 de Introduction to Programming Using Java http://math.hws.edu/javanotes/index.html

Capítulo 8 de Introduction to Programming Using Java http://math.hws.edu/javanotes/index.html

¡Saludos, muy buenas a todos!

Si ya has intentado realizar alguno que otro ejercicio que te dejé en capítulos anteriores o has estado explorando por tu cuenta en maratones de programación (hackathons), es muy probable que te hayas topado con uno que otro error (bug). Entonces en este capítulo vamos a introducirte a un concepto bastante importante en programación, que es el de depuración. Cabe mencionar que esto es simplemente una introducción a depuración, mas no a cómo escribir código de manera robusta y vigorosa.

Depuración básicamente significa resolver alguno que otro error que aparece cuando estamos código que hemos escrito. Existen varios tipos de errores:

Errores al momento de compilar: el código no puede ejecutarse debido a que el proceso de compilación es interrumpido.

```
String oracion = "Vam
oracion.length(); //
oracion.charAt(9) //
```

Errores de sintaxis son fáciles de identificar gracias a los EDIs.

Errores al momento de ejecución de código: son más difíciles de identificar, debido a que en estos errores el código sí compila, sin embargo los egresos del código no son coherentes a lo que el problema especifica. En otras palabras, el programa funciona, pero no resuelve el problema.

```
def ej_por(m):
    for i in range(m):
        print(i ** 2)
```

El código no es robusto: El código funciona y devuelve los resultados tal como lo indica el problema. Sin embargo, el código es incapaz de lidiar con casos extremos.

```java
public static void main(String[] args) {
        System.out.println(Math.sqrt(-100.0)); // NaN
    }
```

```python
import math
error_o_no = math.sqrt(-1)
print(error_o_no)  # math.sqrt(-1)
                   # ValueError: math domain error
print("¡funciona bien!")
```

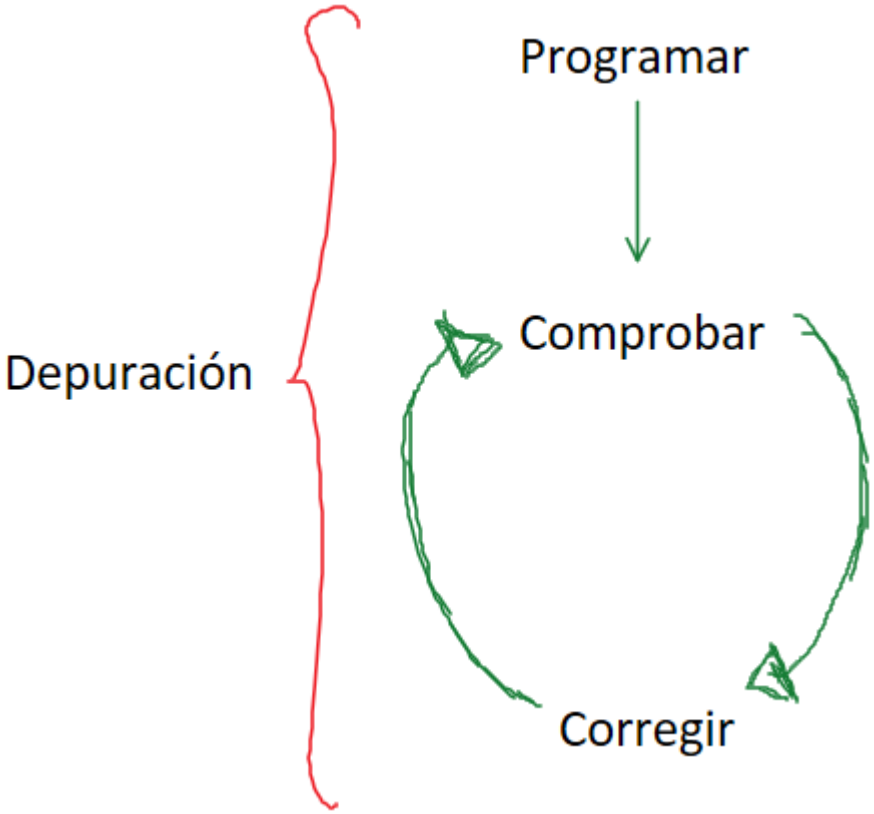
```python
import math
error_o_no = math.sqrt(100)
print(error_o_no)  # math.sqrt(-1)
                   # ValueError: math domain error

print("¡funciona bien!")
```

Es mejor que aparezca un mensaje como “Por favor no uses un número negativo dentro de una raíz par”, ya que permite que el programa siga siendo ejecutado, en vez de crasheó. Por ejemplo, cuando estás navegando en tu buscador web o jugando algún juego, esta aplicación simplemente se cierra de manera inesperada. No es un bug, es simplemente una entrada que no fue manejada de manera correcta.

Entonces hoy vamos a enfocarnos a cómo resolver la segunda clase de problemas, las que ocurren durante ejecución.

La filosofía detrás de depuración es que esta es un proceso iterativo. Observemos el siguiente diagrama:



Vos empiezas con un “borrador” en donde simplemente intentas con algunas líneas de código que crees que resuelve el problema. Luego compruebas que el código ejecute y resuelva el problema. Si hay errores los corriges, y sigues comprobando hasta que el estés satisfecho con los egresos del código.

En Python: Uso de Jupyter Notebook

Personalmente, la manera más rápida de depurar código, por lo menos para tareas cortas y sencillas, es usar Jupyter Notelab. Podemos suponer que es un EDI. Este facilita bastante la escritura, ejecución y depuración de código. Jupyter Notebook no es como VS code, una aplicación local. Esta requiere una conexión a internet. Puedes acceder a un cuaderno Jupyter usando el terminal anaconda ejecutando el comando *jupyter notebook*.

```
In [6]: 'i' * 9
Out[6]: 'iiiiiiiii'
```

```
In [4]: # Dado un entero m, imprime la siguiente estructura. Lo que está entre paréntesis no es necesario imprimir.

def piramide_inv(m):
    reverso = m
    for i in range(1, m + 1):
        print(str(i) * reverso)
        reverso -= 1
```

```
In [5]: piramide_inv(9)

111111111
22222222
3333333
444444
55555
6666
777
88
9
```

```
In [7]: import math
import random
```

Puedes también importar librerías externas. Tal vez tengas que instalar librerías que no vienen preinstaladas.

En Java: Imprimir y creación de archivos para probar el código

Jupyter no funciona con Java, lamentablemente. La manera en la cual personalmente te recomiendo a trabajar con Java con el objetivo de que puedas profesionalizarte es escribir tus propios archivos que prueban el código. Tienes que importar la librería Junit. Puedes visitar http://www.java2s.com/Code/Jar/j/Downloadjunitjar.htm para descargar esta librería externa que contiene funciones útiles para depurar código. También necesitarás descargar otra librería llamada hamcrestcore (no sé para qué se usa): http://www.java2s.com/Code/Jar/h/Downloadhamcrestall13jar.htm#google_vignette

El ejemplo que vamos a usar es revertir un arreglo. ¿Cómo revertir una arreglo? Supongamos que tenemos el siguiente arreglo [1, 3, 9, 3, 1]. Podemos crear un nuevo arreglo de igual tamaño y llenar ese arreglo con un bucle que empieza desde el tamaño del arreglo, y termina en 0. Luego devolvemos este arreglo. ¿Cómo chequeamos que hemos invertido el arreglo? Tenemos 2 opciones por ahora:

1. Imprimir y chequear personalmente los egresos de las funciones con entradas:

```
public static int[] revertirArreglo (int[] miLista) {
        int[] miListaRevertida = new int[miLista.length];
        int indice = miLista.length - 1;
        for (int contador = 0; contador < miLista.length; cont
ador ++ ) {
            miListaRevertida[contador] = miLista[indice ];
            indice -= 1;
        }
        return miListaRevertida;
    }

    public static void main(String[] args) {
        int[] test1 = {2, 3, 4, 5, 2};
        int[] miListaInversa = revertirArreglo(test1);
        System.out.println(Arrays.toString(miListaInversa));
    }
```

2. Crear archivos para probar el código por mi cuenta:

```
import org.junit.Test; //
http://www.java2s.com/Code/Jar/j/Downloadjunitjar.htm
import static org.junit.Assert.assertArrayEquals;
import static proyectosJava.RevertirArreglo.revertir;
public class RevertirArregloTest {

    @Test
    public void testReves(){
        int[] prueba1 = {5, -1, 0, 8, 91};
        int[] prueba1ReversoEsperado = {91, 8, 0, -1, 5};
        int[] prueba1Funcion = revertir(prueba1);
        // chequear si prueba1Funcion y prueba1Reverso son lo
mismo

        assertArrayEquals(prueba1ReversoEsperado, prueba1Funci
on);
    }
    @Test
    public void testReves2() {
        int[] test2 = {0, 0, 0};
        int[] test2Reves = {4, 3, 2, 1, 0};
        assertArrayEquals(test2Reves,test2);
    }

    @Test
    public void testReves3() {
        int[] test3 = {0, 1, 2 ,3, 4};
        int[] test3Reves = {4, 3, 2, 1, 0};
        assertArrayEquals(funcion.revertirArreglo(test3), test
3Reves);
    }

    @Test
    public void testReves4() {
        int[] vacio = {};
        int[] vacioEsperado = {};
        assertArrayEquals(vacioEsperado, funcion.revertirArreg
lo(vacio));
    }
```

```
}
```

Crear archivos para probar el código puede ser tedioso, sin embargo será bastante útil para cuando trabajemos con proyectos más grandes que incluyen mucho más código para resolver problemas más complejos. De hecho, archivos de depuración usualmente son usados en proyectos que involucran inteligencia artificial en donde equipos de programación generan gráficos o tablas para demostrar la eficacia de sus algoritmos comparados con los demás competidores..

Ejercicios

Hoy no vamos a agregar nuevos ejercicios. Te invito a que completes algunos ejercicios anteriores, pero esta vez intento escribir algunos archivos de depuración tanto en Python como en Java. De hecho, en Python también puedes encontrar la función *assert.*

En resumen:

Depuración es corregir tu código para que este ejecute fluidamente y devuelva el egreso esperado. Puedes depurar manualmente o escribir tus propios archivos de depuración.

Textos consultados

Eck, D. J. (2020). *Introduction to Programming Using Java* (8th ed.). Obtenido de http://math.hws.edu/javanotes/index.html

Jara, I., & Hepp, P. (2016). *Enseñar Ciencias de la Computación:Creando oportunidades para los jóvenes de América Latina.* Microsoft.

Zelle, J. (2017). *Python Programming: An Introduction to Computer Science* (3th ed.). FRANKLIN, BEEDLE. Obtenido de http://iran-lms.com/images/images/Books/PDF/Python-Programming_-An-Introduction-to-Computer-Science-Franklin-Beedle--Associates-2016---John-M.-Zelle.pdf