ANyP, Práctica 1b.

Elementos de programación Python: Estructuras de control.

Estructuras de control.

Las estructuras de control permiten especificar el orden en que se ejecutaran las instrucciones de un algoritmo. Todo algoritmo puede diseñarse combinando tres tipos básicos de estructuras de control:

- *secuencial*: las instrucciones se ejecutan sucesivamente una después de otra,
- *de selección*: permite elegir entre dos conjuntos de instrucciones dependiendo del cumplimiento (o no) de una condición,
- *de iteración*: un conjunto de instrucciones se repite una y otra vez hasta que se cumple cierta condición.

Combinando estas tres estructuras básicas es posible producir un flujo de instrucciones más complejo pero que aún conserve la simplicidad inherente de las mismas. La implementación de un algoritmo en base a estos tres tipos de estructuras se conoce como *programación estructurada* y este estilo de programación conduce a programas más fáciles de escribir, leer y modificar.

Estructura secuencial.

La estructura de controlmás simple está representada por una sucesión de operaciones donde el orden de ejecución coincide con la aparición de las instrucciones en el algoritmo (o código del programa). La figura 1 ilustra el diagrama de flujo correspondiente a esta estructura. En Python una estructura secuencial es simplemente un conjunto de sentencias simples, una después de otra.

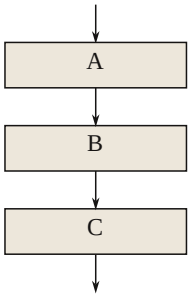


Figura 1: Estructura secuencial.

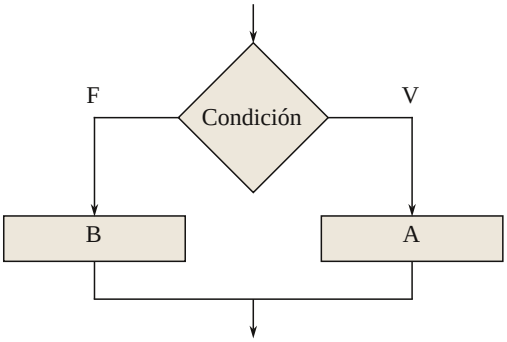


Figura 2: Estructura de selección.

Estructura de selección.

La estructura de selección permite que dos conjuntos de instrucciones alternativas puedan ejecutarse según se cumpla (o no) una determinada condición. El pseudocódigo de esta estructura es descrito en la forma *si . . . entonces . . . sino . . .* , ya que si p es una condición y A y B respectivos conjuntos de instrucciones, la selección se describe como si p es verdadero entonces ejecutar las instrucciones A, sino ejecutar las instrucciones B. Codificamos entonces esta estructura en pseudocódigo como sigue.

Si *condición* **entonces**
  instrucciones para condición verdadera
sino
  instrucciones para condición falsa
fin_si

El diagrama de flujo correspondiente se ilustra en la figura 2. En Python, su implementación tiene la siguiente sintaxis:

```
if condición:
  sentencias para condición verdadera
else:
  sentencias para condición falsa
```

Sangrado ("indentación")

Es importante notar que en Python la indentación (espacios o tabulaciones al inicio de la línea) es crucial, ya que define los bloques de código. El bloque de código que se ejecuta si la condición es verdadera

debe estar indentado con respecto a la sentencia `if`, al igual que el bloque que corresponde al caso en que la condición sea falsa.

La condición en la estructura de selección es especificada en Python por una *expresión lógica*, esto es, una expresión que devuelve un dato de tipo lógico: verdadero (`True`) o falso (`False`). Una expresión lógica puede formarse comparando los valores de expresiones aritméticas utilizando *operadores relacionales* y pueden combinarse usando *operadores lógicos*. El conjunto de operadores relacionales involucra a las relaciones de igualdad, desigualdad y de orden, las cuales son codificadas en Python como se indica en la tabla 1.

Operador	Significado
$<$	menor que
$>$	mayor que
$<=$	menor o igual que
$>=$	mayor o igual que
$==$	igual a
$!=$	distinto a

Tabla 1: Operadores relacionales.

Por otro lado, los operadores lógicos básicos son la *negación*, la *conjunción*, la *disyunción* (inclusiva) y *equivalencia*, cuya codificación en Python se indica en la tabla 2. El operador `not` indica la negación u opuesto de la expresión lógica. Una expresión que involucra dos operandos unidos por el operador `and` es verdadera si ambas expresiones son verdaderas. Una expresión con el operador `or` es verdadera si uno cualquiera o ambos operandos son verdaderos [1].

Operador	Significado
`not`	negación
`and`	conjunción
`or`	disyunción (inclusiva)

Tabla 2: Operadores lógicos.

Cuando en una expresión aparecen operaciones aritméticas, relacionales y lógicas, el *orden de precedencia* en la evaluación de las operaciones es como sigue:

1. Operadores aritméticos.
2. Operadores relacionales.
3. Operadores lógicos, con prioridad: `not`, `and` y `or`.

[1] Estos enunciados no son más que las conocidas tablas de verdad de la lógica matemática

Operaciones que tienen la misma prioridad se ejecutan de izquierda a derecha. Por supuesto, las prioridades pueden ser modificadas mediante el uso de paréntesis.

Ejercicio 1. Dadas las variables con los valores que se indican:

$a = 2.0$, $b = 5.0$, $c = 10.0$, $d = 2.5$, $e = -4.0$, $i = 2$, $j = 3$,$k = -2$, $f = False$ $t = True$,

deducir el valor lógico de cada una de las expresiones lógicas siguientes. Indicar el orden en que se evalúan.

a) `t and f or False`
b) `a**i+b <= b/c+d`
c) `i/j == 2+k and b/c+d >= e+c/d-a**j`
d) `(b*j+3.0) == (d-e) and (not f)`

Ejercicio 2. Implemente una estructura de selección para verificar si un número es negativo o no negativo (positivo o cero).

Ejercicio 3. Las raíces de una ecuación cuadrática $a\,x^2 + b\,x + c = 0$ serán reales o complejas dependiendo del signo del discriminante $\Delta = b^2 - 4\,a\,c$. Implemente una estructura de decisión para determinar la naturaleza de las raíces conocidos los coeficientes de la ecuación.

Ejercicio 4. Implemente una estructura de decisión para determinar si un número entero es par o impar (*Ayuda*: utilice el operador módulo (`%`) el cual devuelve el resto de la división de x por y).

Ejercicio 5. Dada una esfera de radio R, considerando su centro como origen de coordenadas se quiere determinar si un punto de coordenadas (x, y, z) está dentro o fuera de la esfera. Implemente un algoritmo para éste problema.

Ejercicio 6. Considérese en el plano un rectángulo dado por las coordenadas (x_S, y_S) de su vértice superior izquierdo y las coordenadas (x_I , y_I) de su vértice inferior derecho. Se quiere determinar si un punto (x, y) del plano está dentro o fuera del rectángulo. Implemente la solución en un algoritmo.

Ejercicio 7. Dado tres números reales distintos se desea determinar cuál es el mayor. Implemente un algoritmo apropiado (*Ayuda*: considere ya sea un conjunto de estructuras de selección anidadas o bien dos estructuras de selección en secuencia).

Estructuras de selección anidadas.

La sentencia que comienza en el bloque de instrucciones verdadero o falso de la estructura de selección puede ser cualquiera, incluso otra sentencia `if-else`. Cuando ésto ocurre en una o ambas bifurcaciones de la estructura, se dice que las sentencias if están *anidadas*.

En muchas circunstancias se desea ejecutar un conjunto de instrucciones sólo si la condición es verdadera y no ejecutar ninguna instrucción si la condición es falsa. En tal caso, la estructura de control se simplifica, codificándose en pseudocódigo como sigue (y con un diagrama de flujo como se indica en la figura 3)

```
Si condición entonces
  instrucciones para condición verdadera
fin_si
```

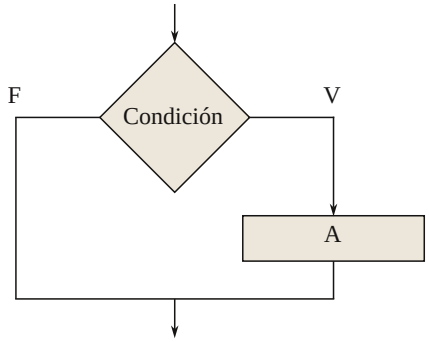


Figura 3: Estructura de selección sin sección /it sino.

La sintaxis correspondiente en Python es,

```
if condición:
  sentencias para condición verdadera
```

Ejercicio 8. Implemente un algoritmo que intercambie los valores de dos números reales si están en orden creciente pero que no realice ninguna acción en caso contrario.

Ejercicio 9. Implementar el cálculo del valor absoluto de un número real con una sentencia `if`.

Otra circunstancia que se suele presentar es la necesidad de elegir entre más de una alternativa de ejecución. En este caso podemos utilizar la estructura multicondicional cuya lógica se puede expresar en la forma *si . . . entonces sino si . . . sino . . .*. El pseudocódigo correspondiente es descrito como sigue (y su diagrama de flujo se ilustra en la figura 4).

```
Si condición 1 entonces
  instrucciones para condición 1 verdadera
sino si condición 2 entonces
  instrucciones para condición 2 verdadera
...
sino si condición N entonces
  instrucciones para condición N verdadera
sino
  instrucciones para todas
 las condiciones falsas
fin_si
```

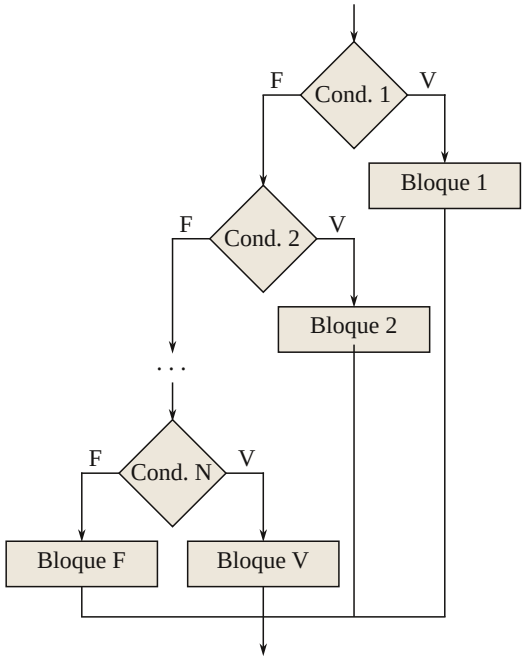


Figura 4: Estructura multicondicional.

Aquí, cada condición se prueba por turno. Si la condición no se se satisface, se prueba la siguiente, pero si la condición es verdadera, se ejecutan las instrucciones correspondientes para tal condición y luego se va al final de la estructura. Si ninguna de las condiciones son satisfechas se ejecutan las instrucciones especificadas en el bloque correspondientes al *sino* final. Debería quedar claro entonces que para que una estructura condicional sea eficiente sus condiciones deben ser /it mutuamente excluyentes. La codificación de esta estructura en Python se indica a continuación.

```
if condición 1:
  sentencias para condición 1 verdadera
elif condición 2:
  sentencias para condición 2 verdadera
  ...
elif condición N:
  sentencias para condición N verdadera
else:
  sentencias para todas condiciones falsas
```

Ejercicio 10. Implementar una estructura multicondicional para la evaluación de la función

$$f(x) = \begin{cases} e^{-x} & \text{si } x < -1, \\ e & \text{si } -1 \leq x \leq 1, \\ e^{x} & \text{si } x > 1. \end{cases}$$

Ejercicio 11. Un examen se considera desaprobado si la nota obtenida es menor que 4 y aprobado en caso contrario. Si la nota está entre 7 y 9 (inclusive)

el examen se considera destacado, y entre 9 y 10 (inclusive) sobresaliente. Implementar un algoritmo para indicar el *estatus* de un examen en base a su nota.

Estructura de iteración.

La *estructura de control iterativa* permite la repetición de una serie determinada de instrucciones. Este conjunto de instrucciones a repetir se denomina *bucle* (*loop*, en inglés) y cada repetición del mismo se denomina iteración. Podemos diferenciar dos tipos de bucles:

- Bucles donde el número de iteraciones es fijo y conocido de antemano.
- Bucles donde el numero de iteraciones es desconocido de antemano. En este caso el bucle se repite mientras se cumple una determinada condición (*bucles condicionales*).

Para estos dos tipos de bucles disponemos de sendas formas básicas de la estructura de control iterativa. Comencemos con un bucle cuyo número de iteraciones es conocido a *priori*. El bucle `for` se utiliza cuando se sabe de antemano cuántas veces se quiere que se ejecute el bloque de código. Generalmente esto se produce sobre una secuencia de elementos (como una lista, tupla, o rango).
Ejemplo de bucle `for` que itera 5 veces:

```
for i in range(5): # i va de 0 a 4
  print(i)
```

El diagrama de flujo correspondiente se ilustra en la figura 5.

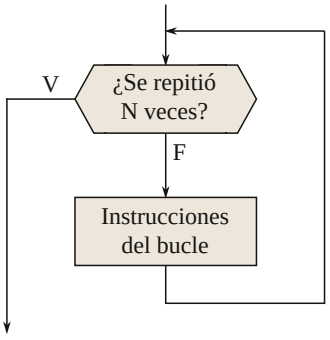


Figura 5: Estructura de iteración.

✍ `i` es el *índice*, una variable que va tomando un valor particular de la secuencia en cada iteración. Naturalmente, no necesita llamarse "`i`".

✍ los valores *inicial* y *final* del índice estarán dados por el primer y el último valor de la secuencia.

✍ El número de iteraciones dependerá del número de elementos de la secuencia.

✍ Dentro de las instrucciones del bucle *es legal* modificar la variable *índice* (en otros lenguajes no lo es), aunque no es aconsejable. Asimismo, al terminar todas las iteraciones el valor del índice del lazo no cambia y queda definido por la última iteración del bucle.

Otra forma de implementar un lazo for es recorriendo una lista:

```
frutas = ["manzana", "banana", "uva"]
for fruta in frutas:
  print(fruta)
```

En este caso, el índice recorre una lista de cadenas de caracteres. El bucle itera sobre cada elemento de la lista y ejecuta el bloque de código (`print(fruta)`), donde `fruta` toma el siguiente valor de la lista en cada iteración, hasta recorrerlos todos.
Otro uso de bucles `for` se da con la *comprensión de listas*, que nos permite crear listas aplicando una expresión a cada elemento de una secuencia, opcionalmente filtrando elementos que cumplen cierta condición. Lo mejor es ver su aplicación con un ejemplo básico:

```
# Genera una lista con el cubo de
# cada elemento de la lista original.
num = [1, 2, 3, 4, 5]
cubo = [x**3 for x in num]
print(cubo)
```

que da como resultado,

```
# [1, 8, 27, 64, 125]
```

Un ejemplo que incluya una condición a cumplir es:

```
num = range(10)
mult_de_3 = [x for x in num if x % 3 == 0]
print(mult_de_3)
```

que mostrará en pantalla la lista:

```
[0, 3, 6, 9]
```

Ejercicio 12. Imprimir una tabla de los cuadrados y cubos de los primeros N números enteros positivos. Ordenar la tabla primero en orden ascendente y luego en orden descendente.

Ejercicio 13. Calcular la suma y multiplicación de los N primeros números enteros positivos,

$$\sum_{i=1}^{N} i, \qquad \prod_{i=1}^{N} i.$$

Inicialización de los acumuladores.

Siempre recordar inicializar a cero la variable que será utilizada para acumular una suma repetida y a uno la variable que acumulará un producto repetido.

Ejercicio 14. Considérese el siguiente conjunto de bucles repetitivos anidados.

```
for i in range(1,6):
  print("Iteracion externa = ",i)
  for j in range(1,5):
    print("Iteracion interna = ",j)
```

¿Cuantas iteraciones del bucle interno se realizan por cada iteración del bucle externo? ¿Cuántas iteraciones se realizan en total?

Ejercicio 15. Tabular la función $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2}$ para x en el intervalo $[0, 1]$ con un paso $h = 0.1$. *Ayuda*: notar que si $[a, b]$ es el intervalo bajo consideración, los puntos donde hay que evaluar f están dados por $x_i = a + i\,h$ con $i = 0, ..., N$, siendo $N = (b - a)/h$).

Ejercicio 16. Generar una lista con los números del uno al veinte utilizando la función `range()`. Utilizando comprensión de listas y la lista recién creada, cree una nueva lista que incluya sólo los elementos pares que sean menores a 10. Muestre el resultado en pantalla.

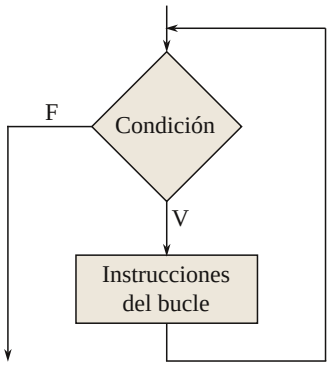


Figura 6: Estructura de iteración condicional.

Consideremos, finalmente, los bucles condicionales. Aquí, el número de iteraciones no es conocido a priori, sino que el bucle se repite *mientras* se cumple una determinada condición. En este caso, su diagrama de flujo se ilustra en la figura 6 y la estructura iterativa se describe en pseudocódigo de la siguiente forma,

Mientras *condición* **hacer:**
instrucciones del bucle

✍ La condición se evalúa antes y después de cada iteración del bucle. Si la condición es verdadera las instrucciones del bucle se ejecutarán y, si es falsa, el control pasa a la instrucción siguiente al bucle.

✍ Si la condición es falsa cuando se ejecuta el bucle por primera vez, las instrucciones del bucle no se ejecutarán.

✍ Mientas que la condición sea verdadera el bucle continuará ejecutándose indefinidamente. Por lo tanto, para terminar el bucle, en el interior del mismo debe tomarse alguna acción que modifique la condición de manera que su valor pase a falso. Si la condición nunca cambia su valor se tendrá un *bucle infinito*, la cual es una situación no deseable.

En Python un bucle condicional se codifica como sigue.

```
while condición:
  sentencias del bloque
```

donde *condición es* una expresión lógica.

Ejercicio 17. Determinar cual es el primer valor de N para el cual la suma de los N primeros números enteros positivos, $\sum_{i=1}^{N} i$, excede a 10 000.

Python, además del `while`, dispone de una estructura más general para bucles condicionales, cuyo diagrama de flujo se representa en la fig 7:

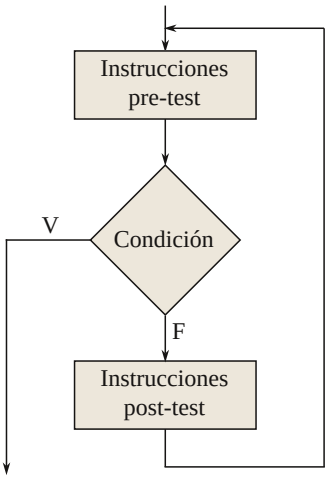


Figura 7: Estructura de iteración condicional general.

y la codificación en pseudocódigo es:

```
Repetir
   instrucciones pre-condición
   Si condición terminar repetir
   instrucciones post-condición
fin_repetir
```

La codificación en Python será de la forma,

```
for
  sentencias del bloque pre-condición.
  if condición:
    break  # Sale del bucle.
  sentencias del bloque post-condición.
```

Implementando lo aprendido.

Los siguientes ejercicios plantean diversos problemas. Diseñar un algoritmo apropiado implementando su pseudocódigo y su diagrama de flujo correspondiente. Luego codificarlo en un programa Python.

Ejercicio 18. Determinar si un año dado es bisiesto o no. Recordar que un año es bisiesto si es divisible por 4, aunque si es divisible por 100 no es bisiesto, salvo si es divisible por 400. Así, 1988 fue bisiesto, como también lo fue el año 2000, pero no 1800.

Ejercicio 19. Dado un conjunto de N números determinar cuántos de ellos son negativos, positivos o cero.

Ejercicio 20. Calcular las soluciones de una ecuación cuadrática $a\,x^2 + b\,x + c = 0$. Contemplar todas las alternativas posibles (raíces reales distintas, iguales y complejas). Testear el programa para el siguiente conjunto de coeficientes:

a) $a = 2$, $b = 2$, $c = -12$ (raíces reales 2 y -3),
b) $a = 2$, $b = 4$, $c = 2$ (raíz doble -1),
c) $a = 1$, $b = 0$, $c = 1$ (raíces conjugadas, i y -i).

Ejercicio 21. Dado un conjunto de N números reales determinar cual es el máximo, el mínimo y la media aritmética del conjunto.

Ejercicio 22. El *máximo común divisor*, mcd, (a, b) de dos número enteros positivos a y b, con $a \geq b > 0$ puede ser calculado por el *Algoritmo de Euclides*, según el cual el mcd es igual al último resto no nulo que se obtiene por aplicación sucesiva de la división entera entre el divisor y el resto del paso anterior. Esto es,

$$\begin{aligned}(a, b) &= (b, r_1)\\ &= (r_1, r_2)\\ &= \dots\\ &= (r_{n-1}, r_n)\\ &= (r_n, 0)\\ &= r_n\end{aligned}$$

Implementar este algoritmo para calcular el máximo común divisor de dos números enteros cualesquiera dados, contemplando la posibilidad de que alguno de ellos, o ambos, sea negativo o cero (*Ayuda*: $(a, b) = (|a|, |b|)$, $(a, 0) = |a|$ y $(0, 0)$ no está definido). Verificar el programa calculando $(25950, 1095) = 15$, $(252, -1324) = 4$.

Ejercicio 23. Calcular el seno de un número x a partir de su serie de Taylor:

$$\operatorname{sen} x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Considerar tantos términos como sean necesarios para que el error cometido en la aproximación finita sea menor que cierta tolerancia prescrita (digamos $\leq 10^{-8}$). *Observación*: Notar que dado un término de la serie, el siguiente se obtiene multiplicando por $-x^2$ y dividiendo por el producto de los dos enteros siguientes. Testear el programa tanto con valores pequeños como con valores muy grandes. Comentar los resultados obtenidos.

Ejercicio 24. Es posible estimar la temperatura expresada en °C a distintas profundidades de la Tierra mediante un modelo simple, que damos para *Corteza*, *Manto*, *Núcleo externo* y *Núcleo interno*:

$$T(z) = \begin{cases} T_0 + G_c\, z & 0 \leq z \leq 35 \\ T_{cm} + G_m\,(z - z_{cm}) & z_{cm} < z \leq 2900 \\ T_{mne} + G_{ne}\,(z - z_{mne}) & z_{mne} < z \leq 5150 \\ T_{neni} + G_{ni}\,(z - z_{neni}) & z_{neni} < z \leq 6370 \end{cases}$$

En esta expresión, $G_c = 25\,C/km$ es el gradiente térmico en la Corteza, $G_m = 0.75\,C/km$ el correspondiente al Manto, $G_{ne} = 1.25\,C/km$ al Núcleo externo y $G_{ni} = 0.25\,C/km$ al Núcleo interno. T_{cm} es la temperatura en la frontera entre Corteza y Manto, y z_{cm} la profundidad de la misma. Similarmente para las demás temperaturas y profundidades identificadas.

Escribir un programa que imprima la profundidad y la correspondiente temperatura, en intervalos de $5\,km$, desde la superficie hasta una profundidad fijada por el usuario. Considerar T_0 como la temperatura ambiente, ingresada también por el usuario.

Si el dato entrado es menor que cero, o mayor que 6370, debe imprimirse un mensaje de error.

Opción: Luego del mensaje de error, o de realizar un cómputo, el programa debe pedir el ingreso de un nuevo dato, y debe terminar al ingresarse un número en particular.

Ejercicio 25. Cuando se manipulan datos de campo ó experimentales es posible definir ciertas cantidades que nos permiten analizar su comportamiento.

Así, si disponemos de N datos x_i, $i = 1, ..., N$, podemos definir las siguientes cantidades.

I- La *media aritmética* μ, llamada también *promedio* o simplemente *media*, definida como,

$$\mu = \frac{1}{N}\sum_{j=1}^{N} x_j$$

II- La *media cuadrática* μ_{RMS}, denominada también *valor cuadrático medio*, cuya expresión es,

$$\mu_{RMS} = \sqrt{\frac{1}{N}\sum_{j=1}^{N} x_j^2}$$

la cual constituye un promedio que no tiene en cuenta el signo de los datos.

III- La *desviación media* D_m y la *desviación estándar* σ, definidas como

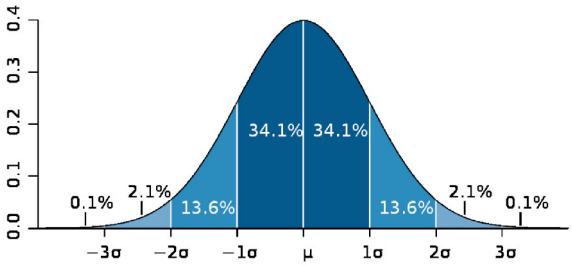
$$D_m = \frac{1}{N}\sum_{j=1}^{N} |x_j - \mu|$$

$$\sigma = \sqrt{\frac{\sum_{j=1}^{N}(x_j - \mu)^2}{N}}$$

respectivamente, las cuales nos dan una idea de cuanto se distancian los datos respecto de su media.

a) Implementar un programa para calcular las cuatro cantidades definidas para un conjunto cualquiera de N datos que ingresará el usuario.

b) Por otra parte, se dice que un conjunto de datos tiene una *distribución gaussiana* si están agrupados alrededor de la media de manera que el porcentaje del número total de datos en cada uno de los ocho intervalos del siguiente gráfico es próximo al indicado.



Implementar un programa para decidir si un conjunto de N datos proporcionados por el usuario sigue una distribución gaussiana. Para simplificar, considere los 6 intervalos entre -3σ y 3σ.

c) Aplique los programas desarrollados a los siguientes datos, los cuales son un subconjunto de las profundidades (medidas en km) de los epicentros de los 437 sismos ocurridos en la Tierra el día 5 de abril de 2010 [2].

$$P = \{0.0, 10.0, 12.1, 12.3, 0.7, 9.4, 6.8, 26.5, 1.9, 7.0, 13.2, 17.4, 9.3, 8.1, 10.3, 0.1, 5.6, 10.0, 10.0, 14.1, 0.4, 217.1, 5.4, 2.9, 7.2, 16.0, 2.3, 7.8, 2.6, 10.0, 3.9, 0.7, 1.6, 12.8, 12.9, 0.0, 4.4, 12.8, 10.1, 0.5, 8.8, 0.6, 4.7, 11.4, 3.3, 12.5, 1.6, 18.9, 6.9, 7.0, 2.5, 7.5, 98.3, 1.1, 6.0, 6.0, 1.4, 9.9, 0.1, 10.4, 0.0, 30.7, 14.2, 2.2, 13.1, 5.7, 2.9, 3.9, 0.7, 1.6, 12.8\}$$

[2] Fuente: USGS (http://earthquake.usgs.gov/).