Universidad Andrés Bello

Facultad de Ingeniería
Asignatura INS121
Semestre 2019-1

PROGRAMACIÓN 1

Sesión 6: Lenguaje de Programación Python PARTE 1

Universidad Andrés Bello

TEMAS SESIÓN 6

Temas de esta Sesión:

- Valores, variables y tipos de datos Python
- Operadores y expresiones Python.
- Conversiones de tipos de datos
- Estructuras secuenciales
- Estructuras condicionales
- Estructuras repetitivas
- Ejercicios Propuestos

VARABLES, VALORES Y TIPOS DE DATOS PYTHON

Variables y Valores

- Para poder escribir programas es necesario almacenar valores en variables para que dichos valores se puedan recordar y usar en cualquier momento que se requiera.

  nombre_variable = expresión

- Ejemplos:

```
edad = 20          # aquí la variable llamada edad está almacenando el valor 20
edad = 30          # ahora la variable edad tiene valor 30. El 20 se perdió
edad = 30 + 10     # ahora edad tiene valor 40. El 30 también se perdió
aumento = 5        # se crea una variable  llamada aumento y guarda el valor 5
edad = edad + aumento   #¿ahora qué valor almacena edad?
```

- Para mostrar el valor de una variable se usa *print*

  ***print**(edad)*

- Para leer el valor de una variable (ingresado por el usuario) se usa *input*

  edad = ***input**(“Ingrese el valor de la edad”)*

- Al ***input*** se debe anteponer el **TIPO DE DATOS**, **visto más adelante**. ***Por defecto el input recibe un string***

Nombres de Variables

- Los nombres de variables deben seguir algunas reglas:
  1) Deben empezar con una letra o un _
  2) Después de la primera letra pueden seguir letras, números o _
  3) No utilizar caracteres especiales (#,&,$,*,etc.)
  4) No se deben utilizar las PALABRAS RESERVADAS de Python
- Palabras reservadas:

```
and       del       from      nonlocal  while
as        elif      global    not       with
assert    else      if        or        yield
break     except    import    pass      True
class     exec      in        raise     False
continue  finnaly   is        return    None
def       for       lambda    try
```

Tipos de Datos y Operaciones

Los tipos de datos básicos o tipos de datos primitivos usados en Python son:

- Un tipo de datos para representar números enteros llamado ***int***.
- Un tipo de datos para representar números reales llamado ***float***.
- Un tipo de datos que representa letras o texto (string) llamado ***str***.
- Un tipo de datos que representa valores lógicos o booleanos llamado ***bool***.

Tipos de Datos: *int*

- El tipo de datos ***int*** representa valores de números enteros positivos y negativos. Las variables que son de este tipo de datos permite almacenar valores tales como: número del mes actual, el año actual, la edad de una persona, cantidad de alumnos que asisten a la clase de Programación 1, cantidad de personas del grupo familiar de Pepito, etc.

```
>>> type(20)
<class 'int'>
>>>
```

- Leer un número y almacenarlo en una variable entera:

edad = ***int*** (***input***(“**Ingrese la Edad de la persona** "))

Tipos de Datos: *float*

- El tipo de datos ***float*** representa valores de números reales (con decimales) positivos y negativos. Las variables que son de este tipo de datos permite almacenar valores con precisión tales como: mi nota en programación 1, mi peso actual, mi estatura, etc.

```
>>> type(2.35)
<class 'float'>
>>> type(2.0)
<class 'float'>
>>>
```

- Leer un número y almacenarlo en una variable real:

estatura = ***float*** (***input***("**Ingrese la estatura de la persona**"))

Tipos de Datos: *bool*

- El tipo de datos ***bool*** representa booleanos o de lógica binaria. Las variables que son de este tipo de datos permite almacenar 2 posibles valores ***True*** o ***False (empiezan con mayúscula)*** y se utilizan cuando se quiera responder una pregunta con un “Si” o con un “no” (¿vas a la universidad hoy?, ¿me escuchas?, ¿vas a la fiesta el viernes?, etc.).

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

```
>>> type(true)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    type(true)
NameError: name 'true' is not defined
>>>
```

- Leer una pregunta y almacenarlo en una variable booleana. El usuario debe ingresar ***True*** o ***False*** :

pregunta = ***bool*** (***input***(“**¿Quieres ir al cine?**"))

Tipos de Datos: *str*

- El tipo de datos ***str*** representa secuencias o cadenas de texto o caracteres (o un string). Las variables que son de este tipo de datos permite almacenar valores que tengan cualquier texto: mi nombre, el nombre de mi canción preferida, lugar donde fui de vacaciones, etc. Estos valores deben ir escritos entre comillas simples o con comillas dobles.

```
>>> type("Programación 1")
<class 'str'>
>>>
```

- Leer un texto y almacenarlo en una variable string. El usuario debe ingresar el valor entre comillas (simples o dobles):

nombre = ***str*** (***input***(“**Ingrese el nombre de la asignatura**"))

ó

nombre = ***input***(“**Ingrese el nombre de la asignatura**")

OPERADORES Y EXPRESIONES

Tipos de Operadores

- Operadores aritméticos
- Operadores de comparación
- Operadores lógicos
- Operadores de texto

Operadores Aritméticos en Python

- **Operadores Aritméticos**. Las expresiones que usan operadores aritméticos entregan como resultado un valor de tipo ***int*** o ***float*** según corresponda

OPERADOR	NOMBRE OPERADOR	EJEMPLO EXPRESIÓN ARITMÉTICA	RESULTADO
+	Suma	5+3	8
-	Resta	5-3	2
*	Multiplicación	5*3	15
/	División real	5/3	1.6666666666666667
//	División entera	5//3	1
**	Exponenciación	5**3	125
%	Módulo	5%3	2

Operadores de Comparación en Python

- **Operadores de Comparación**. Las expresiones que usan operadores de comparación siempre entregan como resultado un valor de tipo booleano (***True*** o ***False***)

OPERADOR	NOMBRE OPERADOR	EJEMPLO EXPRESIÓN DE COMPARACIÓN	RESULTADO
<	Menor	5<3	False
<=	Menor o igual	5<=3	False
>	Mayor	5>3	True
>=	Mayor o igual	5>=3	True
!=	Distinto	5!=3	True
==	Igualdad	5==3	False

Operadores Lógicos en Python

- **Operadores Lógicos**. Las expresiones que usan operadores lógicos siempre entregan como resultado un valor de tipo booleano (***True*** o ***False***) y se aplican a expresiones que usan operadores de comparación.

OPERADOR	NOMBRE OPERADOR	EJEMPLO EXPRESIÓN LÓGICA	RESULTADO
not	Negación	**not** 5<3	True
and	Conjunción lógica (Y)	5<=3 **and** 5!=3	False
or	Disyunción lógica (O)	5==3 **or** 5!=3	True
^	Disyunción exclusiva	5<=3 **^** 5==3	False

```
x       y       x or y
---------------------
False   False   False
False   True    True
True    False   True
True    True    True

x       y       x and y
---------------------
False   False   False
False   True    False
True    False   False
True    True    True

x       not x
------------
False   True
True    False

x       y       x ^ y
---------------------
False   False   False
False   True    True
True    False   True
True    True    False
```

Operadores Lógicos en Python

- Las tablas de verdad expresan:
  - ✓El resultado de ***or*** es ***verdadero*** cuando cualquiera de los operandos lo es.
  - ✓El resultado de ***and*** es ***verdadero*** solo cuando ambos operandos lo es.
  - ✓El operador ***not*** invierte el valor del operando.
  - ✓La o exclusiva, **^**, es ***verdadera*** cuando uno y solo uno de los operandos lo es.

```
x       y       x or y
---------------------
False   False   False
False   True    True
True    False   True
True    True    True

x       y       x and y
---------------------
False   False   False
False   True    False
True    False   False
True    True    True

x       not x
-------------
False   True
True    False

x       y       x ^ y
---------------------
False   False   False
False   True    True
True    False   True
True    True    False
```

Precedencia y asociatividad de los operadores en Python

- La precedencia se aplica al escribir expresiones con más de un operador. En este caso se debe considerar lo siguiente:
  - ✓ Las expresiones se evalúan según la ***precedencia*** de los operadores (especifica el orden de las operaciones)
  - ✓ Operaciones con igual precedencia se resuelven por orden de ***asociatividad*** (especifica si un operando se agrupa con el de su izquierda o el de su derecha).
  - ✓ Los operadores de comparación tienen menor precedencia que los operadores aritméticos.
  - ✓ Los operadores lógicos tienen un orden estricto:
    - ❖ Primero se evalúa el ***not***
    - ❖ Segundo se evalúa el ***and***
    - ❖ Tercero se evalúa el ***or***

Precedencia y asociatividad de los operadores en Python

- La precedencia y asociatividad se indica a continuación:

	Operador	Asociatividad	Ejemplo		
1	`**`	$\longleftarrow$	$2**3**2$	$= 2**(3**2)$	$= 512$
2	`+, -` (unarios)	-	$-2**2$	$= -(2**2)$	$= -4$
3	`*, /, //, %`	$\longrightarrow$	$15/3*2$	$= (15/3)*2$	$= 10$
4	`+, -` (binarios)	$\longrightarrow$	$3-4+5$	$= (3-4)+5$	$= 4$
1	`<, <=, >=, >, >=, !=, ==`	$\longrightarrow$	`3<4<=4<5`	`= (((3<4)<=4)<5)`	`= True`
2	`not`	-	`not not 5>2`	`= (not not) 5>2`	`= True`
3	`and`	$\longrightarrow$	`not True and False`	`= (not True) and False`	`= False`
4	`or`	$\longrightarrow$	`True or True and False`	`= True or True and False`	`= False`

Operadores de texto en Python

- La Concatenación (+): Junta dos string
- La Repetición (*a): Repite el string ***a*** veces (***a*** debe ser un número entero)

OPERADOR	NOMBRE OPERADOR	EJEMPLO EXPRESIÓN DE TEXTO	RESULTADO
+	Concatenación	“Adiós ” **+** “mundo cruel”	“Adiós mundo cruel”
*a	Repetición	“JA” ***4**	“JAJAJAJA”

CONVERSIONES DE TIPO DE DATOS

Conversiones de tipo de datos

- Es necesario tener algunos cuidados al operar con tipos de datos que son distintos entre sí.

- En general con los tipos numéricos no hay problemas. Ejemplos:
  - ✓ Si multiplicamos un entero y un float, obtenemos un float:
  - ✓ Si dividimos un entero por un float, también obtenemos un float:

```
>>> 5*0.5
2.5
```

```
>>> 5/0.5
10.0
```

- Pero, si concatenamos un string con el resultado de un cálculo (es decir con un número) pasa lo siguiente. Ej:

```
>>> "Son las "+ (3 + 15)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    "Son las "+ (3 + 15)
TypeError: can only concatenate str (not "int") to str
```

  - ✓ El compilador envía error que dice que no puede convertir un entero a un string **implícitamente**, no así cuando operamos entre **int** y **float** ya que Python convierte internamente el **int** a un **float**.
  - ✓ Cuando el primer operando es un string, Python intenta hacer una concatenación y no una suma. Este tipo de problemas se resuelve haciendo una ***conversión explícita*** a la variable o al resultado de la expresión que corresponda. En este caso sería convertir explícitamente al resultado de la expresión numérica en un string, es decir: >> ”Son las ” + **str**(3+12)
  - ✓ ¿Qué resultado arroja las siguientes expresiones?:
    - >> (4) + (5)
    - >> **str**(4)+ **str** (5)

Conversiones de tipo de datos

- No sólo podemos convertir strings, podemos también por ejemplo:
  - ✓ Anteponer ***int*** a una expresión y con eso le estamos diciendo a Python que el resultado de la expresión lo entregue como entero.
  - ✓ También podemos convertir un string a un entero pero sólo en el caso en que dicho string esté compuesto sólo por números (sin puntos ni comas).

```
>>> int("3")
3
>>> int("3.0")
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    int("3.0")
ValueError: invalid literal for int() with base 10: '3.0'
```

  - ✓ Lo mismo podemos hacer para obtener valores de tipo ***float*** si aplicamos la conversión a un string que posee sólo valores enteros o valores enteros con decimales.

```
>>> float("3")
3.0
>>> float("3.0")
3.0
```

ESTRUCTURAS SECUENCIALES EN PYTHON

Estructuras Secuenciales

- En Python: cada una de las instrucciones están separadas por un salto de línea
- No hay llaves, corchetes ni palabras claves explícitas como en otros lenguajes de programación.
- El único delimitador es el símbolo dos puntos (:) y el propio indentado del código.
- Los bloques de instrucciones van definidos por su sangrado (o tabulación).
- Un “bloque de instrucciones” corresponde a una sección de instrucciones que define funciones, sentencias if, bucles for, while, etc.
- El sangrado comienza un bloque y su ausencia lo termina. Esto quiere decir que el espacio en blanco es significativo y debe ser consistente.
- El sangrado puede ser con tabulador o cuatro o tres o dos espacios, incluso un espacio. El único requisito es que sea consistente con las instrucciones del mismo bloque.

ESTRUCTURAS CONDICIONALES EN PYTHON

Estructuras Condicionales

- Las estructuras condicionales se utilizan para ***alterar el flujo de ejecución*** de los programas con el fin de:
  - ✓Tomar decisiones a partir de los datos y/o resultados intermedios y, en función de éstas, ejecuten ciertas sentencias y otras no.
  - ✓Tomar decisiones a partir de los datos y/o resultados intermedios y, en función de éstas, ejecuten ciertas sentencias más de una vez.
- El primer tipo de ***alteración del flujo de control*** se efectúa con sentencias condicionales o de selección y el segundo tipo con sentencias iterativas o de repetición.

Estructura condicional simple: if

```
if condiciones:
    instrucciones R.condición True
    .....
```

```
a = float(input("ingrese a: "))
b = float(input("ingrese b: "))
if a != 0:
    x = -b/a
    print "x es", x
```

```
num = int(input("Ingrese un número positivo menor que 100: "))
if num<0 or num>=100:
    print("ERROR: Ingrese un número positivo menor que 100 !!")
print("Ha ingresado el número: ", num)
```

- Las condiciones son expresiones compuestas por expresiones comparativas y/o lógicas cuyo resultado es un True o un False. Las condiciones se evalúan siempre. Si el resultado es:
  - ✓ ***True*** se ejecuta el bloque de instrucciones con sangrado bajo del **if**
  - ✓ ***False*** no se ejecuta dicho bloque de instrucciones.
- Si existe más de una condición, éstas son unidas a través de un operador lógico ***and*** u ***or***
- Es importante señalar que el bloque de instrucciones que se ejecuta cuando la condición es verdadera **debe ir sangrado**, puesto que Python utiliza el sangrado para reconocer las líneas que forman un bloque de instrucciones.

Estructura condicional doble: if-else

```
if condiciones:
    instrucciones R.condición True
    ...
else
    instrucciones R.condición False
    ...
```

```
a = float(input("ingrese a: "))
b = float(input("ingrese b: "))
if a != 0:
    x = -b/a
    print "x es", x
else:
    print "no tiene solución"
```

```
edad = int(input("¿Cuántos años tiene? "))
if edad< 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
print("¡Hasta la próxima!" )
```

- El ***else*** se agrega cuando se requiere ejecutar otro conjunto de instrucciones cuando NO se cumpla la condición
- Si el resultado es:
  - ✓ ***True*** se ejecuta el bloque de instrucciones sangrado bajo del **if**
  - ✓ ***False*** se ejecuta el bloque de instrucciones sangrado bajo el **else**

Sentencia condicional múltiple

```
if condiciones:
    instrucciones R.condición True
    ...
else:
    if otras_condiciones:
        instrucciones R.condición False y R.otras_condiciones True
        ........
.......
```

```
edad = int(input("¿Cuántos años tiene? "))
if edad<18 and edad>0:
    print("Es usted menor de edad")
else:
    if edad <=0:
        print("Error: Edad ingresada < que 0")
    else:
        print("Es usted mayor de edad")
print("¡Hasta la próxima!" )
```

- A veces, cuando la condición NO se cumple (en el ***else***) podría ser necesario agregar otra condición.
- En Python existe ***elif***, donde nos ahorramos una indentación:

```
if condiciones:
    instrucciones R.condición True
    ...
elif otras_condiciones:
        instrucciones R.condición False y R.otras_condiciones True
        ........
.......
```

Estructura condicional con
Operador Ternario

- Es un operador que toma tres argumentos. Se utiliza cuando no se indican instrucciones específicas para la condición verdadera.

if condiciones else instrucciones R.condición ***False***

- Ejemplo:

```
a = float(input("ingrese a: "))
b = float(input("ingrese b: "))
print -b/a
if a != 0 else print "no tiene solución"
```

ESTRUCTURAS ITERATIVAS EN PYTHON

Estructuras Repetitivas

- Las estructuras repetitivas se utilizan cuando se requiere ejecutar varias veces un mismo conjunto de instrucciones mientras se cumpla una condición.
  - ✓Permite volver a solicitar datos al usuario cuando éste se equivoca (***validar***). Ej: el usuario se equivoca en ingresar una opción de menú o cuando ingresa una edad menor que cero, etc.
  - ✓***Optimizar*** memoria cuando se requiere solicitar la misma variable varias veces. Ej: leer la edad de 10 personas, leer el precio de 5 productos, etc.
  - ✓Permite ***acumular*** valores asociados a una misma característica para después mostrar su resultado y/o realizar algún cálculo posteriormente. Ej: sumar la edad de 5 personas para después calcular el promedio de las edades, calcular la cuenta total del supermercado, etc.
  - ✓Permite ***contar*** situaciones que cumplan una condición para informar al usuario o para realizar algún cálculo. Ej: contar las personas que sean mayor de edad, calcular el promedio de las edades de personas que son mayores de edad.

Estructura iterativa while

```
while condiciones:
    instrucciones R.condición True
    ........
```

```
contCiclo = 1
suma = 0
while (contCiclo<=5 ):
    precio = int (input("Precio del producto " + str(contCiclo) + ": "))
    suma = suma + precio
    contCiclo = contCiclo +1
print("Monto total de la compra: ", suma)
```

- Se evalúa la condición (o condiciones) y si el resultado es:
  - ✓ ***True*** se ejecuta el bloque de instrucciones con sangrado bajo del **while**. **Este conjunto de instrucciones se repetirá 0 o más veces.**
  - ✓ ***False*** NO se ejecuta las instrucciones del while.
- Por lo tanto, la condición del ***while*** se ejecuta (o evalúa) antes de ejecutar las instrucciones que contiene dicho *while*.
- Por lo general, la estructura ***while*** requiere de un ***contador*** para controlar la cantidad de veces que se está ejecutando el bloque de instrucciones pertenecientes al *while*, por ejemplo ***contCiclo*** del ejemplo de arriba.
- En el ejemplo de arriba, a parte de ocupar el contador ***contCiclo,*** usa el acumulador ***suma,*** que suma el precio de todos los productos ingresados.

Estructura iterativa while

Diga qué imprime cada programa. Realice el seguimiento de cada variable

```
#PROGRAMA 1
i = 0
while i < 10:
    print(i)
    i = i + 2 #i+=2
print(i)
```

```
#PROGRAMA 2
i = 1
while i < 10:
    print(i)
    i = i + 2 #i+=2
print("")
print(i)
```

```
#PROGRAMA 3
i = 0
while i <= 10:
    print(i)
    i = i + 2 #i+=2
print("\n",i)
```

```
#PROGRAMA 4
i = 100
while i <= 10:
    print(i)
    i += 1 #i=i+1
print(i)
```

```
#PROGRAMA 5
i = 100
while i >= 10:
    print(i)
    i += 1 #i=i+1
print(i)
```

Estructura iterativa for - in

```
for variable in serie_de_valores :
    instrucciones para cada valor de la serie
    ........
```

- Esta estructura se lee de la siguiente forma: para todo elemento (o valor) dentro de la ***serie_de_valores*** **hacer** el conjunto de instrucciones del ciclo ***for***
- Permite recorrer una serie de elementos o valores, es decir, en cada iteración la ***variable*** toma el valor correspondiente a la ***serie_de_valores***
- Las instrucciones contenidas dentro del ciclo ***for*** se ejecutarán tantas veces como elementos o (valores) existan en la ***serie_de_valores***.

```
for nombre in ["Pepe", "Ana", "Juan"]:
    print("Hola,", nombre + ".")
```

Salida

```
Hola, Pepe.
Hola, Ana.
Hola, Juan.
```

Estructura iterativa for – in

Función range

- La estructura repetitiva ***for*** en muchas ocasiones utiliza la función ***range*** ya que es una función que nos entrega una serie de valores que nos permitiría ***iterar***.

`range(número)`	`range(inicio, número)`	`range(inicio, número, paso)`
Genera una serie entre 0 y *número*.	Genera una serie entre *inicio* y *número*.	Genera una serie entre *inicio* y *número* avanzando un *paso*.

```
for i in range(3):
    print(i)
print("Hecho")
```

Salida

```
0
1
2
Hecho
```

```
for i in range(5, 10):
    print(i)
print("Hecho")
```

Salida

```
5
6
7
8
9
Hecho
```

```
for i in range(10, 2, -2):
    print(i)
print("Hecho")
```

Salida

```
10
8
6
4
Hecho
```

Modificadores de Ciclos: *break* y *Continue*

- Los modificadores de ciclos se utilizan para modificar el comportamiento natural de los ciclos.
- ***break***: permite interrumpir un ciclo, independiente de su condición de término.
- ***continue***: permite hacer un salto en el ciclo y continuar con la siguiente iteración.

```
for i in range(1, 10):
    if i%3 == 0:
        print("El número",i," es divisible en 3")
        break
    print(i)
print("Terminé")
```

Salida Python

```
1
2
El número 3 es divisible en 3
Terminé
```

```
for i in range(1, 10):
    if i%3 == 0:
        print("El número",i," es divisible en 3")
        continue
    print(i)
print("Terminé")
```

Salida Python

```
2
El número 3 es divisible en 3
4
5
El número 6 es divisible en 3
7
8
El número 9 es divisible en 3
Terminé
```

Ejercicios Propuestos

1) Evaluar manualmente las siguientes expresiones:
   a) 10+10+10∗10
   b) (3 + 5//4 − 2) − 2∗∗4 + 3∗(7 − 2)

2) Escribir un programa para cada caso siguiente. NO use sentencias repetitivas:

   a) Resolver ecuaciones de segundo grado, que son de la forma:

   $$ax^2 + bx + c = 0$$

   b) Realizar un programa que escriba la calificación correspondiente a una nota, de acuerdo con el siguiente criterio:

0 - 39	Reprobado
40 - 49	Aprobado
50 - 59	Notable
60 - 69	Sobresaliente
70	Matrícula de honor

   c) Que pida dos números enteros y que calcule su división, escribiendo si la división es exacta o no. Tenga en cuenta que no se puede dividir por cero.

   d) Escriba un programa que pida los coeficientes de una ecuación de primer grado (a x + b = 0) y escriba la solución. Se recuerda que una ecuación de primer grado puede no tener solución, tener una solución única, o que todos los números sean solución. Se recuerda que la fórmula de las soluciones es x = -b / a

   e) Escriba un programa que pida un año y que escriba si es bisiesto o no. Se recuerda que los años bisiestos son múltiplos de 4 y 400, pero los múltiplos de 100 no lo son.

Ejercicios Propuestos

3) Escribir un programa para cada caso siguiente. Use sentencias repetitivas con o sin modificadores de ciclo en el caso que corresponda :

a) Escriba un programa que calcule el promedio de 4 notas ingresadas por el usuario. Valide el ingreso de las notas.

b) Escriba un programa que pida al usuario un entero de tres dígitos, y entregue el número con los dígitos en orden inverso. Valide la entrada de datos. Ejemplo:

```
Ingrese numero: 345
543
```

c) El riesgo de que una persona sufra enfermedades coronarias depende de su edad y su índice de masa corporal:

	edad < 45	**edad ≥ 45**
IMC < 22	bajo	medio
IMC ≥ 22	medio	alto

El índice de masa corporal es el cuociente entre el peso del individuo en kilos y el cuadrado de su estatura en metros. Por ejemplo, si una persona pesa 85 [kg] y mide 1.8 [m], su IMC es:

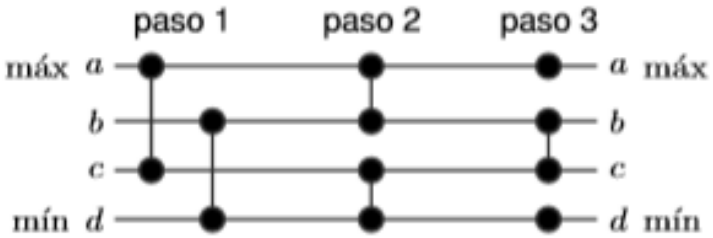
$$\text{IMC} = \frac{85}{(1.8)^2} = \frac{85}{3.24} = 26.23$$

Construya un programa que reciba la estatura, el peso y la edad de una persona, y muestre por pantalla su condición de riesgo, junto con su índice de masa corporal. Valide las entradas de datos.

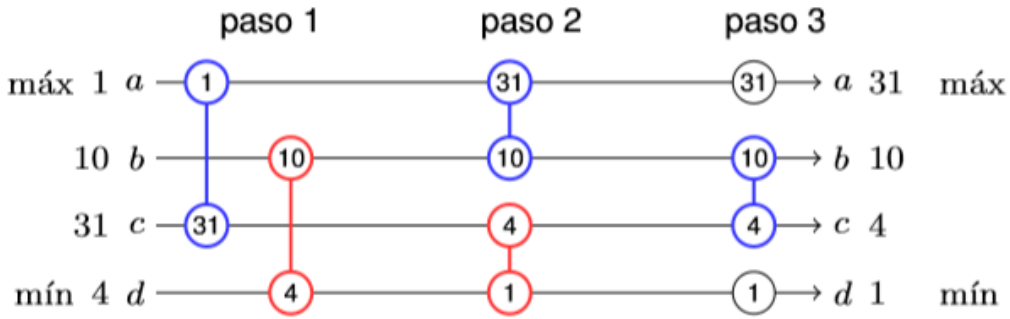
d) Un número N con 4 dígitos es un número ***doble-cuadrado*** cuando éste iguala la suma de dos números al cuadrado: uno formado por los dos primeros dígitos de N y el otro formado por los dos últimos dígitos de N. Por ejemplo, 1233 es un número ***doble-cuadrado*** ya que $1233 = 12^2 + 33^2$.
Construya un programa que reciba un número N e imprima por pantalla un 1 si es doble-cuadrado o en caso contrario imprima un 0. Valide las entradas de datos.

Ejercicios Propuestos

e) Las redes de ordenamiento pueden ser implementadas en hardware o software como una secuencia de elementos comparadores. Por ejemplo, en la siguiente figura se muestra un ejemplo de red para una secuencia de 4 elementos.



Construya un programa que solicite cuatro números a, b, c, d ∈R y los ordene mediante la red de ordenamiento presentada anteriormente. Un ejemplo se muestra a continuación:



Ejercicios Propuestos

f) Los tres lados a, b y c de un triángulo deben satisfacer la desigualdad triangular: cada uno de los lados no puede ser más largo que la suma de los otro dos.
Construya un programa que reciba como entrada los tres lados de un triángulo e indique si el triángulo es válido o no. Valide las entradas.

g) En las competencias de clavados, cada salto es evaluado por siete jueces. Cada juez entrega una puntuación en una escala de 1 a 10, con incrementos de 0.5. La puntuación más alta y la más baja son eliminadas (si se repite valores máximos y/o mínimos se elimina cualquiera de los repetidos). La suma de los cinco puntajes restantes es multiplicada por 0.6, y el resultado es multiplicado por el grado de dificultad del salto. El valor obtenido es el puntaje total del salto.
Construya un programa que lea el grado de dificultad del salto y los 7 puntajes de los jueces, para luego mostrar por pantalla el puntaje total salto. Un ejemplo se muestra a continuación:

```
Grado de dificultad: 3.0
Juez 1: 5.0
Juez 2: 5.5
Juez 3: 4.0
Juez 4: 5.0
Juez 5: 4.5
Juez 6: 5.5
Juez 7: 5.0
El puntaje total es 45.0
```

Ejercicios Propuestos

h) Un edificio tiene 20 pisos de 8 departamentos cada uno. La dueña del edificio ha definido una estrategia para ponerle precio a cada departamento.
El número que identifica cada departamento se divide en dos partes: los dos últimos dígitos indican en qué posición está (de acuerdo al diagrama), y los restantes el piso. Por ejemplo, el departamento 1105 está en el undécimo piso, en la posición 5.
Los dos departamentos al extremo derecho tienen vista al mar, y los dos al extremo izquierdo tienen vista al cerro. Todos los departamentos del primer piso cuestan 100, y todos los departamentos del último piso cuestan 400. Para los pisos intermedios, se ha fijado un precio base de 250; el precio de los departamentos con vista al mar se aumentará en un 15%, y los con vista al cerro se rebajará en un 20%.
Construya un programa que reciba el número del departamento y muestre por pantalla el precio.

i) Los números ***Feos***, son aquellos números que son factorizados por 2, 3 o 5. La secuencia: 1,2,3,4,5,6,8,9,10,12,15 son los primeros 11 números feos. Por convención se incluye el número 1. Realice un programa en Python que entregue la secuencia de los n números feos (n ingresado por teclado).

Ejercicios Propuestos

j) Una pelota de tenis es lanzada desde una altura de H metros. Cada vez que la pelota rebota alcanza una altura máxima equivalente a 5/7 de su altura anterior, siempre y cuando no exista ningún roce. En el caso de existir un roce, la altura alcanzada es similar, es decir, 5/7 de su altura anterior pero menos un 6%. Cuando la pelota alcanza una altura de rebote inferior a los 3 centímetros (0,03 metros) deja de rebotar. Realice un programa Python que entregue cuántos rebotes alcanza a dar una pelota de tenis lanzada desde una altura H considerando sin roce y con roce. Por ejemplo, si lanzo una pelota desde 10 metros de altura, los rebotes serán se indican en la tabla siguiente:

ALTURA SIN ROCE	ALTURA CON ROCE	NRO. DEL REBOTE
10,00000	10,00000	0
7,14286	6,71429	1
5,10204	4,50816	2
3,64431	3,02691	3
2,60308	2,03235	4
1,85934	1,36458	5
1,32810	0,91622	6
0,94865	0,61518	7
0,67760	0,41305	8
0,48400	0,27733	9
0,34572	0,18621	10
0,24694	0,12503	11
0,17639	0,08395	12
0,12599	0,05636	13
0,08999	**0,03784**	**14**
0,06428	0,02541	15
0,04591		16
0,03280		**17**
0,02343		