SEAS
CAMPUS SEAS

3 UNIDAD DIDACTICA

Python

3. Estructuras de control

ÍNDICE

OBJETIVOS

- Estudiaremos las selecciones, que son estructuras que fuerzan la ejecución de una secuencia u otra en función del resultado que tome una función booleana (condición).
- Conoceremos las distintas formas existentes de iteraciones, en las que varía el número de veces que se ejecuta una secuencia en función del resultado que tome una función booleana (es decir, de que se cumpla una condición, llamada de continuación).
- Como objetivo fundamental de esta unidad, dominaremos estas el de dominar estas importantes estructuras, para lo que se realizarán bastantes ejercicios de programación con niveles de dificultad crecientes y se presentarán los algoritmos de fuerza bruta.

INTRODUCCIÓN

La principal característica de una computación imperativa es que se trata de una secuencia de acciones que se ejecutan una tras otra. Si esta secuencia siempre fuese fija, los computadores no se hubiesen desarrollado como lo han hecho, pues se trataría de un recurso por sí solo insuficiente. Las instrucciones ejecutadas siempre serían las mismas independientemente de los valores iniciales del algoritmo, lo que no permitiría resolver problemas que exijan tomar decisiones en función de tales valores iniciales.

Por consiguiente, la secuencia temporal de acciones que se ejecuta no es normalmente idéntica a la secuencia textual de instrucciones. Esto se conoce como “Estructura de control” y es el fundamento del fenomenal éxito del software.

La secuencia de acciones que se ejecutan está determinada por “estructuras de control” que representan:

1. Selecciones (ejecuciones condicionales). Se ejecuta una u otra porción de programa en función de que se cumpla una condición.
2. Iteraciones (repeticiones). Se repite un bloque de programa en función de que se cumpla una condición.

Como una estructura de control puede gobernar a otras estructuras, se habla de “Composición estructurada”. Los lenguajes con composiciones estructuradas se conocen como “Lenguajes estructurados”.

Con este tipo de estructuras se estará en condiciones de escribir programas que resuelvan problemas de un cierto nivel. De hecho, en principio poco más se necesitará para escribir programas, aunque ya desde los comienzos de la informática se vió el interés de las sentencias virtuales que permitan escribir funciones más o menos complejas y que se llamaron rutinas o procedimientos. Se verán en la próxima unidad.

3.1. Sentencias condicionales

Vamos a introducir una estructura nueva para hacer más flexibles nuestros programas. Consideremos que tenemos dos variables x e y, y que una tercera variable z debe tomar el valor máximo de x e y. Está claro que z tiene que ser x o y, pero a priori no sabemos cuál de los dos. Una forma natural de decidirlo sería utilizar una nueva construcción que evaluase cual es mayor mediante una expresión lógica de la forma (x<y) de la que depende la instrucción que se vaya a ejecutar y permitiese elegir entre z=x o z=y de forma conveniente. La notación de la nueva construcción tendrá una forma tal como:

- si x>y: z=x # como x es mayor que y, se asigna a z el valor de x.
- si no: z=y # como x no es mayor que y, se asigna a z el valor de y.

Esto es expresa diciendo que si x>y, se asigna a z el valor de x. En caso contrario, se esigna a z el valor de y.

En general, una selección puede tomar unas de las tres formas siguientes en Python:

`if B1:` `  S1`	Si se cumple la condición B1 (expression lógica con valor True) se ejecuta la secuencia S1, en caso contrario (valor False), no hace nada.

`if B1:`
`  S1`
`else:`
`  S2`
	Si se cumple la condición B1 se ejecuta la secuencia S1, en caso contrario se ejecuta S2.

`if B1:`
`  S1`
`elif B2:`
`  S2`
`else:`
`  S3`
	Si (if) se cumple la condición B1 se ejecuta la secuencia S1. Si no, si (elif) se cumple B2 se ejecuta S2. Si no se cumple ninguna (else), se ejecuta S3.
	elif es una contracción de else if (si no, si).

Vamos a escribir el código para calcular los valores que toma una función matemática para cada valor x de abcisas, codificada en Python. Se trata de la función Signo. Esta función vale 1 para x>0, vale -1 para x<0 y vale 0 para x=0. Por tanto, se define:

$$Sig(x) = \begin{cases} +1 \ \forall x > 0 \\ 0 \ \text{para} \ x = 0 \\ 1 \ \forall x < 0 \end{cases}$$

El código quedaría algo así como:

```
>>>x=0.43
>>>if x > 0:
  y = 1
elif x < 0:
  y = -1
else:
  y = 0
```

EJEMPLO

El siguiente ejemplo determina si una variable x toma valor par o impar. Para ello se divide entre 2. Si el resto es cero, es par. En caso contrario, es impar.

```
>>> x = 9
>>> if x % 2 == 0:
  print ("x es par")
else:
  print ("x es impar")
```

EJEMPLO

Selecciones embebidas. Es posible que haya una selección dentro de otra. En el siguiente ejemplo se contempla esta posibilidad.

```
>>> x = 9
>>> if x>0:
      print (“mayor que cero”)
      if x % 2 == 0:
            print (“ y par “)
            if x>100:
                  print (“ y mayor que 100”)
            else:
                  print (“ y menor o igual que 100”)
      else:
             print(“ e impar”)
             if x>100:
                   print (“ y mayor que 100”)
             else:
                   print (“ y menor o igual que 100”)
   else:
       print(“menor que cero”)
```

Así como algunos lenguajes marcan los bloques con algún carácter de inicio y final, en Python es la indentación de las líneas la que marca los distintos bloques que hay en cada secuencia. Esto hace que la escritura de programas sea más rápida. Los bloques de instrucciones se asocian a una línea de encabezamiento que termina en dos puntos. En el ejemplo anterior, se han representado en colores al objeto de que se comprenda mejor su funcionamiento.

3.2. Composición iterativa

Frecuentemente hay que repetir una secuencia un cierto número de veces, que es función posiblemente de cierto estado inicial. Las composiciones iterativas responden a algoritmos cuya duración dependa del estado inicial. En ocasiones esta composición se conoce también como bucle ó iteración. Para resolver este problema, adoptaremos la estructura:

```
mientras que B:
    S
```

Donde B es una condición, llamada "condición de continuación" de la estructura iterativa, es decir, una expresión lógica (predicado) y S es una secuencia. Mientras la condición B toma valor *True*, se ejecuta la secuencia S. Evidentemente dentro de S tiene que modificarse el valor que toma la condición B para que la iteración termine cuando deba.

Así, al principio de la ejecución, se comprueba la condición B y si toma valor *True* (verdad) se ejecuta la instrucción S, usualmente conocida como cuerpo de la iteración; esto se repite hasta que la condición B no se cumple. Caso de que de entrada no se cumpla la condición, el bloque S no se ejecuta.

La estructura iterativa básica en Python tiene la forma:

```
while B:
    S
```

EJEMPLO

3.1. En el siguiente ejemplo se scriben los números del 1 al 10, su cuadrado y su cubo.

```
>>> x = 0
>>> while x < 10:
        x = x+1
        print (x, x**2, x**3)
```

Los sucesivos valores que va tomando x son:

`x = 0`
`while x<10:`
`  x = x+1`
`print(x, x**2, x**3)`
	Inicialmente x=0
	x<10 por lo que se entra en el bucle de la iteración
	x toma el valor 1 (x=1)
	Escribe:
	1
	1
	1

`while x<10:`
`  x = x+1`
`print(x, x**2, x**3)`
	Como x<10, se ejecuta la iteración
	x toma el valor 2 (x=2)
	Escribe:
	2, 4, 8

`while x<10:`
`  x = x+1`
`print(x, x**2, x**3)`
	Como x<10, se ejecuta la iteración
	x toma el valor 3 (x=3)
	Escribe:
	3, 9, 27

`while x<10:`
`  x = x+1`
`print(x, x**2, x**3)`
	Como x<10, se ejecuta la iteración
	X toma el valor 4 (x=4)
	Escribe:
	4
	16
	64
	
	Y se sigue así hasta que x tome el valor 9. Escribiría
	9
	81
	729
	Entraría en la iteración ya que 9<10. Se incrementaría tomando x el valor 10 y se escribiría:
	10
	100
	1000
	Aquí terminaría la iteración ya que como x es 10 ya no se cumple que x<10

EJEMPLO

3.2. Producto de enteros. Dados dos enteros X e Y no negativos, diseñar un algoritmo que compute en la variable prod el producto XY. Usar como instrucción primitiva la suma.

Como ya sabemos, multiplicar x por y es sumar y veces x. Por tanto, una forma de resolver el problema es plantear una iteración de la forma:

```
prod=x+x+...+x  (y veces).
```

Así, en Python quedaría:

```
x, y = 5, 3
prod=0                  # aún no se ha sumado ninguna x
while y!=0:
       prod=prod+x
       y=y-1
```

Es decir, cada vez que sumamos a la variable prod el valor x, disminuye la variable y en una unidad. Así hasta que y pase a valer cero, en cuyo caso hemos sumados ya y veces, el valor de x.

Veámos como actúa el algoritmo con dos números concretos: si x=5 e y=3 se obtiene lo que a continuación se detalla:

Tras la ejecución de las dos primeras líneas queda.	`x=5, y=3, prod=0`
Tras esto, como y≠0, se entra en la iteración while y se ejecuta: prod=prod+x y=y-1	`prod=0+5=5, x=5, y=2`
Como y≠0, de nuevo se ejecuta el cuerpo de la iteración: prod=prod+x y=y-1	`prod=5+5=10, x=5, y=1`
Como y≠0, de nuevo se ejecuta el cuerpo de la iteración: prod=prod+x y=y-1	`prod=10+5=15, x=5, y=0`
y como y=0, ya se ha terminado. Se observa que prod es el producto de x e y.	

EJEMPLO

3.3. Factorial de un número. Como sabemos, el factorial de un número se define en forma recursiva de la siguiente forma: el factorial de 0 es 1. El factorial de n es n multiplicado por el factorial de n-1. Por ejemplo:

n=0, su factorial es 1.

n=1, su factorial es 1 por factorial de 0=1.

n=2, su factorial es 2 por 1=1.

n=3, su factorial es 3 por 2 por 1=6.

Por tanto, el código para determinar el factorial de un número n es:

```
n = 5
fact = 1
k = n
while k!=0:
    fact = fact * k
    k = k - 1
print("El factorial de
", n, " es:", fact)
```

En la variable **fact** vamos a almacenar los productos parciales, de forma que su valor final será el factorial buscado. Se le da el valor inicial 1 porque si n es cero, no se ejecuta la iteración y por tanto contiene ya la solución al factorial de cero.

El factorial de 5 es 120.

3.3. Otras composiciones iterativas

La iteración *for* en Python es algo distinta de la correspondiente de lenguajes como C o Java. En Python esta sentencia itera sobre los elementos de una secuencia (lista, rango o cadena de caracteres) en el orden en que aparecen en la secuencia.

Operando con listas. Si se considera una lista. Si se considera una secuencia: sec=[s1, s2, s3], esta iteración se escribe en Python:

```
sec=[s1, s2, s3]
for s in sec:
        opera_sobre_la_secuencia
```

Así, cuando comienza la iteración s toma el valor s1 y se ejecuta el bucle de la iteración operando sobre la secuencia. Tras eso, s toma el valor s2 y se vuelve a ejecutar. Finalmente s toma el valor s3 y se vuelve a ejecutar, terminando así la iteración.

EJEMPLO

```
sec=[1, 3, 5, 7]
for s in sec:
      print(s)
1
3
5
7
>>>
```

Inicialmente s toma el valor 1 y se imprime, a continuación s toma el valor 3 y se imprime, a continuación toma el valor 5 y se imprime y finalmente, s toma el valor 7 y se imprime.

- **break y continue:** estas dos sentencias permiten modificar el comportamiento de una iteración utilizando una selección *if* para detenerla si se cumple una cierta condición. Por ejemplo:

```
sec=[s1, s2, s3]
for s in sec:
     if s>0:
                    break
```

break hace que termine la iteración.

Por el contrario, la sentencia continue simplemente se salta un paso de la iteración, saltando al siguiente. Por ejemplo;

```
sec=[s1, s2, s3]

for s in sec:

     if s==3:

          continue
```

Operando sobre rangos, hay una sentencia que tiene una forma más similar a la de otros lenguajes. Así, considerando un rango de enteros:

```
for x in range(1, 12):

  secuencia
```

Asigna internamente el valor 1 al inicio. Cuando se ejecuta "secuencia" automáticamente x pasa a valer 2 y de nuevo se ejecuta la secuencia. Así hasta que en total la secuencia se ejecuta 11 veces.

La iteración *for* puede contener una sentencia *else* que se ejecuta cuando termina la iteración. Por ejemplo:

```
  for x in range(5):

       print(x)

  else:

       print("Ha terminado la iteración")
```

3.4. La visibilidad de las variables y las estructuras de control

Python no restringe la visibilidad de una variable al interior del bloque en el que fue asignada. En el siguiente ejemplo, las variables *visibleAfuera visibleIter y visibleAdentro* están disponibles dentro y fuera del bucle.

```
# visibilidad
visibleAfuera = 5
for visibleIteracion in range(1,3):
    visibleDentro = 9
    print(visibleAfuera)
    print(visibleIteracion)
    print(visibleDentro)
print(visibleAfuera)
print(visibleIteracion)
print(visibleDentro)
```

```
5
1
9
5
2
9
5
2
9
```

3.5. Aplicación de las estructuras de control a los tipos de datos compuestos

En el apartado 3.3 ya se han aplicado las iteraciones *for* a listas y rangos. En este apartado se va a profundizar la aplicación de las estructuras de control a tipos mutables compuestos, tales como listas y diccionarios para su creación y para el desarrollo de algoritmos de búsqueda, fusión y conversión de listas a diccionarios, así como para añadir elementos a un diccionario.

Vamos a comenzar con las listas. En primer lugar, vamos a escribir el código para saber si un determinado elemento está contenido en una lista usando el operador in.

```
fruta = [“plátano”, “manzana”, “pera”, “melón”]

if “pera” in fruta:

     print(“Pera es una fruta”)
```

- Fusión es añadir a los elementos de una lista los pertenecientes a otra lista. Veamos la fusión de una lista voc2 en una lista voc1.

```
voc1 = [‘a’, ‘e’, ‘i’, ‘o’, ‘u’]

voc2 = [‘a’, ‘e’, ‘i’, ‘o’, ‘u’]

for voc in voc2:

     voc1.append(voc)

print(voc1)
```

  Esta notación no es habitual ya que se usa en programación orientada a objetos que se estudiará más adelante. Se puede interpretar diciendo que se manda el mensaje *append* (añadir) a la lista voc1. La variable voc2 es un parámetro.

Veamos con los diccionarios.

- Creación de un diccionario que contenga como valores el cuadrado de los índices comenzando por el índice 1. En primer lugar, se crea un diccionario vacío.

```
# creación de un diccionario que contenga el cuadrado
de los índices más uno (comenzando por el uno)
n = int(input(“Escribe un número: “))
d = dict()
for x in range(1, n+1):
    d[x]= x**2
```

  Supongamos que se introduce el número 6. El diccionario quedaría:

  {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}

- Adición de un nuevo par <clave>:<valor>, 10:100

  d.update({10: 100}) # añadimos un elemento al diccionario

  Con esto, el *script* queda en la forma siguiente. Si se introduce el número 5:

```
n = int(input("Escribe un número: "))
d = dict()
for x in range(1, n+1):
    d[x]= x**2
d.update({10: 100}) # añadimos un elemento al
diccionario
print(d)
```

  {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 10: 100}

  En la línea d.update({10: 100}) de nuevo nos encontramos con una notación típica de programación orientada a objetos que se puede interpretar como que se envía el mensaje update al objeto d.

- Conversión de dos listas (una lista para las claves y otra para los valores) en un diccionario. Se usará la función **zip** que permite iterar sobre múltiples listas (en general, contenedores) simultáneamente.

```
# conversión de dos listas en un diccionario
claves = ['1', '2', '3']
valores = ['10', '20', '30']
d = dict(zip(claves, valores))
print(d)
```

  {'1': '10', '2': '20', '3': '30'}

  Se usa **zip** para correlaciones de claves y valores.

- Trabajamos con un diccionario en el que el campo valor está constituido por otro diccionario.

```
diccionario = {'primero':{'a':'PrimeroA',
'b':'PrimeroB', 'c':'PrimeroC'},
'segundo':{'a':'SegundoA', 'b':'SegundoB'}}
for clave in diccionario:
    print(clave)
    for valor in diccionario[clave]:
        print(valor, ':', diccionario[clave][valor])
```

primero

a : PrimeroA

b : PrimeroB

c : PrimeroC

segundo

a : SegundoA

b : SegundoB

3.6. Programa ejemplo

Vamos a practicar introduciendo un programa sencillo. Lo ejecutaremos y posteriormente se introducirán errores deliberadamente para ver lo que sucede. El programa realizará un cambio de moneda entre tres monedas: euro, libra y dólar. Nos pedirá que introduzcamos la moneda que tenemos y queremos cambiar, la cantidad que queremos cambiar y la moneda que queremos adquirir. Veamos el código que se ha escrito en la ventana del editor:

```
print("CAMBIO DE MONEDA")
CD_E = 0.88      # conversión de dólar a euros 1 dólar = 0,88
euros
CE_L = 0.94       # conversión de euro a libra 1 euro = 0.94
libras
CD_L = 0.83      # conversión de dólar a libra 1 dólar = 0,83
libras
cambio = True
while (cambio == True):
    print("Introduce las monedas en minúsculas (libra, dólar,
euro)")
    x = input("¿Qué modena tienes?:")
    x1 = int(input("¿Cuántas quires cambiar?:"))
    y = input("¿A qué moneda quieres cambiar?:")
    if x == "libra":
        pluralx = "libras"
        if y == "dólar":
            y1 = (1/CD_L)*x1
            pluraly = "dólares"
        else:
            pluraly = "euros"
            y1 = (1/CE_L)*x1
    if x == "dólar":
        pluralx = "dólares"
        if y == "libra":
            y1 = CD_L*x1
           pluraly = "libras"
        else:
            y1 = CD_E*x1
            pluraly = "euros"
    if x == "euro":
        pluralx = "euros"
        if y == "libra":
            y1 = CE_L*x1
            pluraly = "libras"
        else:
            y1 = (1/CD_E)*x1
            pluraly = "dólares"
    print("Los ", x1, pluralx, "son", y1, pluraly)
    print("¿Otro cambio? (si/no):")
    c = input()
    if c == "si":
        cambio = True
    else:
        cambio = False
```

Figura 3.9. Programa de cambio de modeda.

Vemos que es un programa muy elemental, pero que sirve para aprender a escribir y testear programas.

En la figura 3.2 se muestra el mismo programa pero escrito de otra forma. Se deja al alumno que vea cuál es la versión más legible.

```
print("CAMBIO DE MONEDA")
CD_E = 0.88     # conversión de dólar a euros 1 dólar = 0,88
euros
CE_L = 0.94      # conversión de euro a libra 1 euro = 0.94
libras
CD_L = 0.83     # conversión de dólar a libra 1 dólar = 0,83
libras
cambio = True
while (cambio == True):
    print("Introduce las monedas en minúsculas (libra, dólar,
euro)")
    x = input("¿Qué modena tienes?:")
    x1 = int(input("¿Cuántas quieres cambiar?:"))
    y = input("¿A qué moneda quieres cambiar?:")
    if (x == "libra") & (y == "dólar"):
            y1 = (1/CD_L)*x1
            pluralx, pluraly = "libras", "dólares"
        elif (x== "libra")& (y== "euro"):
            y1 = (1/CE_L)*x1
            pluralx, pluraly = "libras", "euros"
        elif (x== "dólar")& (y== "libra"):
            y1 = CD_L*x1
            pluralx, pluraly = "dolares", "libras"
        elif (x== "dólar")& (y== "euro"):
            y1 = CD_E*x1
            pluralx, pluraly = "dolares", "euros"
        elif (x== "euro")& (y== "libra"):
            y1 = CE_L*x1
            pluralx, pluraly = "euros", "libras"
        elif (x== "euro")& (y== "dólar"):
            y1 = (1/CD_E)*x1
            pluralx, pluraly = "euros", "dólares"
    print("Los ", x1, pluralx, "son", y1, pluraly)
    print("¿Otro cambio? (si/no):")
    c = input()
    if c == "si":
        cambio = True
    else:
        cambio = False
```

Figura 3.10. Otra versión del programa de cambio de modeda.

Sobre este programa, se debe:

- Leer detenidamente el programa hasta comprender su funcionamiento.
- Escribir el programa en la ventana del editor. No se debe guardar el programa ya que se almacena automáticamente en memoria cuando se ejecuta.
- Para ejecutar el programa se puede ir a Run en el menú principal o pulsar sobre el triángulo superior derecho de color verde.

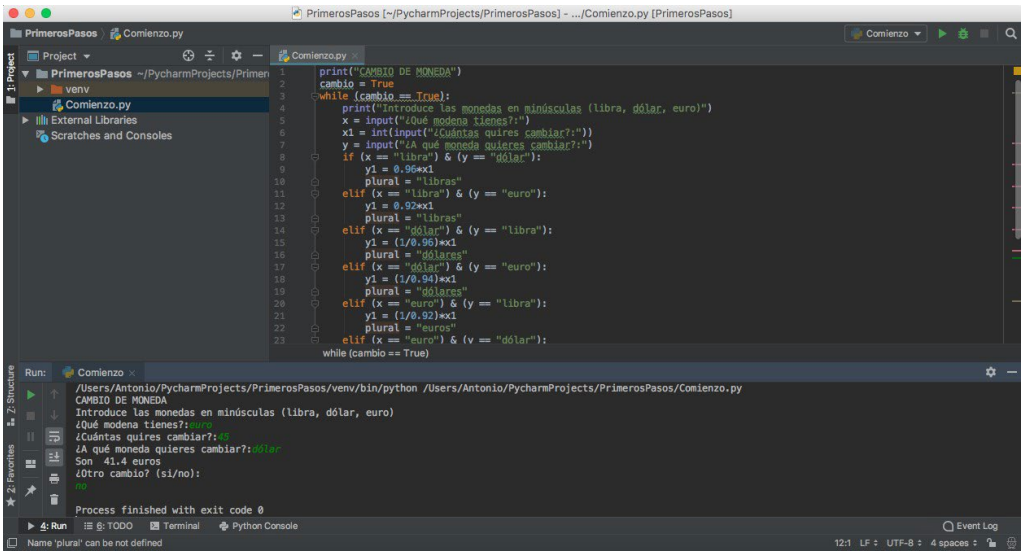
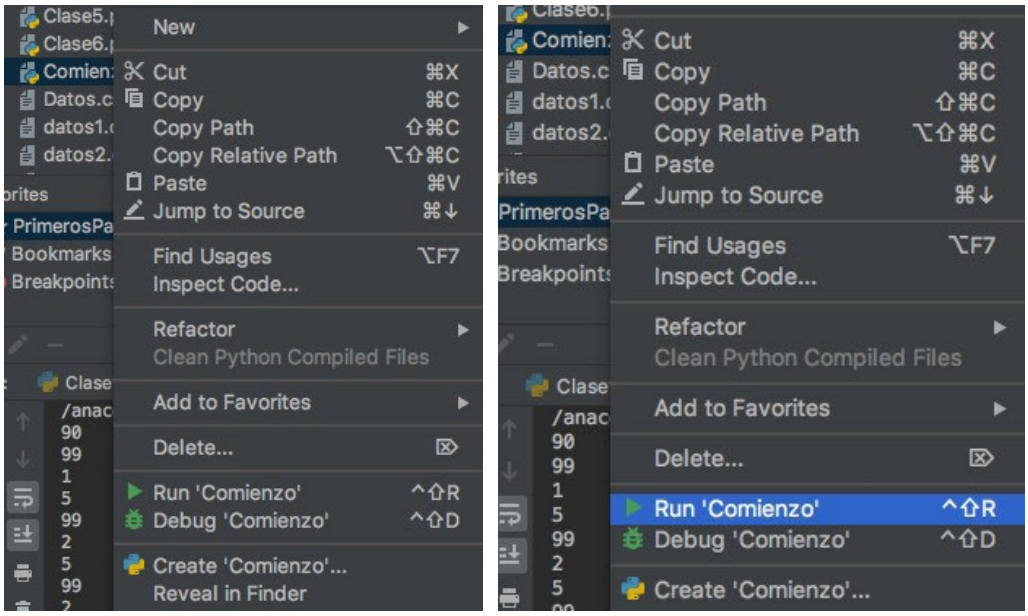
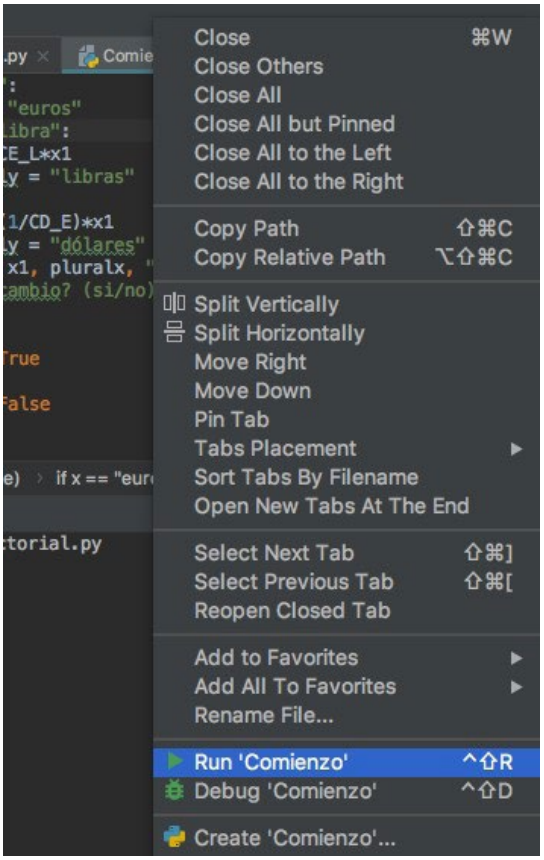


Figura 3.11. Editor con el script de cambio de moneda.

- Para ejecutar el programa, nos situamos sobre su identificador en la ventana superior izquierda y con el botón derecho del ratón pulsamos sobre Run <Identificador>.



- Otra forma es situarnos sobre la solapa que contiene el nombre del *script* y pulsar el botón derecho del ratón. Nos aparecerá un desplegable en el que figura Run <identificador>



En la figura 3.3, se muestra el programa en el editor, el triángulo verde que da lugar a la ejecución del programa cuando se pulsa sobre él, y el resultado de la ejecución en la región inferior de la pantalla. A su vez, en la figura 3.4, se muestra en detalle dicha región inferior.

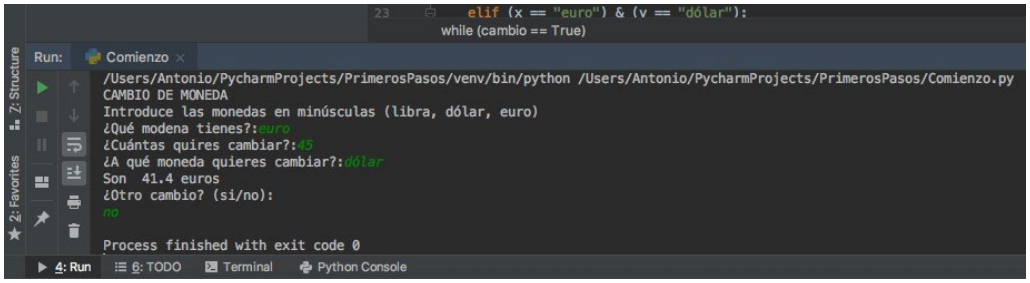


Figura 3.12. Zona inferior de salida de datos.

Tras la ejecución se ha introducido como moneda a cambiar el euro, por un importe de 45 euros. Como moneda que se desea el dólar. El resultado ha sido que los 45 euros se convierten en 51,13 dólares. La respuesta a si se desea otro cambio ha sido que no, lo que da lugar a la terminación del programa. Dado que se ha ejecutado con éxito, aparece la leyenda: *Process finished with exit code 0*.

3.7. Cuestiones

- Volver a ejecutar el programa, pero esta vez escribiendo “si” a la pregunta de si deseamos otro cambio. ¿Qué sucede?.

  CAMBIO DE MONEDA

  ¿Otro cambio? (si/no):

  si

  Introduce las monedas en minúsculas (libra, dólar, euro)

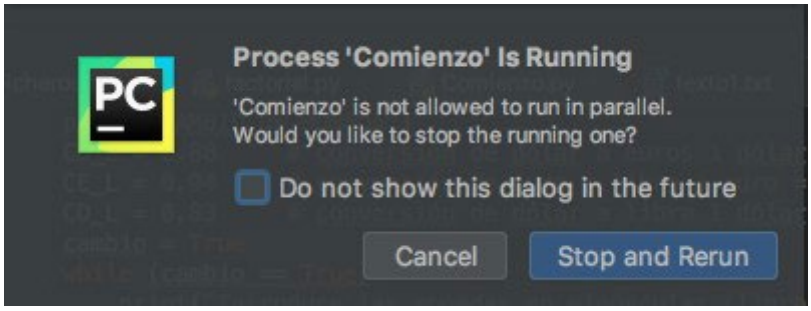
  ¿Qué moneda tienes?:

- ¿Qué ha sucedido con el triángulo verde?.

  Como el programa no ha terminado se ha transformado en:

- Pulsar el nuevo símbolo verde que aparece en su lugar. ¿Qué sucede?

  Nos aparece en pantalla por si queremos detener el programa y volver a ejecutarlo.



Vamos a practicar un poco introduciendo errores para ver lo que sucede:

- Tras una de las sentencias if, quitar los dos puntos y volver a ejecutar el programa.

```
    if (x == “libra”) & (y == “dólar”)
                                      ^

SyntaxError: invalid syntax

Process finished with exit code 1
```

- Quitar el *int* que precede a la sentencia *input* en la que se pide que se introduzca la cantidad que se desea cambiar. ¿Qué sucede?.

```
CAMBIO DE MONEDA

Introduce las monedas en minúsculas (libra, dólar, euro)

¿Qué moneda tienes?:dólar

¿Cuántas quieres cambiar?:300

¿A qué moneda quieres cambiar?:euro

Traceback (most recent call last):

   File "/Users/Antonio/PycharmProjects/PrimerosPasos/
Comienzo.py", line 29, in <module>

    print("Los ", x1, pluralx, "son", y1, pluraly)

NameError: name 'pluralx' is not defined

Process finished with exit code 1
```

- En la sentencia x == "dólar" de la línea 13, quitar uno de los caracteres de igualdad. ¿Qué sucede?

```
CAMBIO DE MONEDA

Introduce las monedas en minúsculas (libra, dólar, euro)

¿Qué moneda tienes?:dólar

¿Cuántas quieres cambiar?:300

¿A qué moneda quieres cambiar?:euro

Traceback (most recent call last):

   File "/Users/Antonio/PycharmProjects/PrimerosPasos/
Comienzo.py", line 29, in <module>

    print("Los ", x1, pluralx, "son", y1, pluraly)

NameError: name 'pluralx' is not defined

Process finished with exit code 1
```

- ¿Qué sucede si se introduce dólar sin el acento?

```
    if (x = "libra") & (y == "dólar"):
            ^
SyntaxError: invalid syntax

Process finished with exit code 1
```

- ¿Qué sucede si cuando se pide: ¿Cuántas quieres cambiar?, se introduce por error: dólar?

```
Introduce las monedas en minúsculas (libra, dólar, euro)
¿Qué modena tienes?:dólar
¿Cuántas quires cambiar?:200
¿A qué moneda quieres cambiar?:euro
Traceback (most recent call last):
  File "/Users/Antonio/PycharmProjects/PrimerosPasos/
Comienzo.py", line 29, in <module>
    print("Los ", x1, pluralx, "son", y1, pluraly)
NameError: name 'pluralx' is not defined

Process finished with exit code 1
```

- ¿Qué sucede si en la línea:

```
CD_E = 0.88 # conversión de dólar a euros 1 dólar = 0,88
euros
```

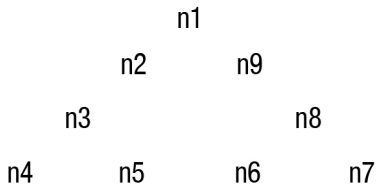
se sustituye el punto por una coma?

```
CAMBIO DE MONEDA
Introduce las monedas en minúsculas (libra, dólar, euro)
¿Qué modena tienes?:euro
¿Cuántas quires cambiar?:200
¿A qué moneda quieres cambiar?:dólar
```

```
Traceback (most recent call last):

   File  "/Users/Antonio/PycharmProjects/PrimerosPasos/
Comienzo.py", line 27, in <module>

    y1 = (1/CD_E)*x1

TypeError: unsupported operand type(s) for /: 'int' and
'tuple'

Process finished with exit code 1
```

3.8. Algoritmo de fuerza bruta

Diseñar un algoritmo que sitúe los nueve primeros enteros en los tres lados de un triángulo, a razón de cuatro números por lado. Encontrar las distintas combinaciones que hacen que la suma de los tres lados sea la misma. Supondremos los números situados en la forma:



En la figura 3.5 aparece el código que implementa el algoritmo solución del problema propuesto.

```
#  n1 a n9 son los números, suma es la suma de un lado, contador cuenta las soluciones
que cumplen la condición
contador = 0
for n1 in range (1,10):
  for n2 in range(1,10):
    if n1!=n2:
      for n3 in range (1,10):
        if (n3!=n1) and (n3!=n2):
          for n4 in range (1,10):
            if (n4!=n1) and (n4!=n2) and (n4!=n3):
              suma = n1+n2+n3+n4     # suma de los números del primer lado.
              # Comienza el segundo lado
              for n5 in range (1,10):
                if (n5!=n1) and (n5!=n2) and (n5!=n3) and (n5!=n4):
                  for n6 in range (1,10):
                    if (n6!=n1) and (n6!=n2) and (n6!=n3) and (n6!=n4) and (n6!=n5):
                      for n7 in range(1,10):
                         cumple1 = (n7!=n1) and (n7!=n2) and (n7!=n3) and (n7!=n4) and
                         n7!=n5) and (n7!=n6)
                        if (cumple1) & (suma==n4+n5+n6+n7):
                           for n8 in range(1,10):
                           cumple2 = (n8!=n1) and (n8!=n2) and (n8!=n3) and (n8!=n4)
                        cumple3 = (n8!=n5) and (n8!=n6) and (n8!=n7):
                              if cumple2 and cumple3:
                                for n9 in range(1,10):
                                cumple4=(n9!=n1) and (n9!=n2) and (n9!=n3) and (n9!=n4)
                         cumple5 = (n9!=n5) and (n9!=n6) and (n9!=n7) and (n9!=n8)
                                  if (cumple4) and (cumple5) and (suma==n7+n8+n9+n1):
                                       print("Secuencia: ", n1, " ", n2, " ", n3, " ",
                                       n4, " " , n5, " ", n6, " ", n7, " ", n8, " ", n9)
```

3.9. Escritura de *scripts*

Vamos a escribir el programa anterior (*script*), usando un editor sobre el que escribiremos el *script* para posteriormente ejecutarlo. Es la figura 3.13 aparece editado el programa anterior. Veamos como se hace:

```
#  n1 a n9 son los números, suma es la suma de un lado, contador cuenta las soluciones que cumplen la condición
contador = 0
for n1 in range (1,10):
  for n2 in range(1,10):
    if n1!=n2:
      for n3 in range (1,10):
        if (n3!=n1) and (n3!=n2):
          for n4 in range (1,10):
            if (n4!=n1) and (n4!=n2) and (n4!=n3):
              suma = n1+n2+n3+n4    # suma de los números del primer lado. Comienza el segundo lado
              for n5 in range (1,10):
                if (n5!=n1) and (n5!=n2) and (n5!=n3) and (n5!=n4):
                  for n6 in range (1,10):
                    if (n6!=n1) and (n6!=n2) and (n6!=n3) and (n6!=n4) and (n6!=n5):
                      for n7 in range(1,10):
                        cumple1 = (n7!=n1) and (n7!=n2) and (n7!=n3) and (n7!=n4) and (n7!=n5) and (n7!=n6)
                        if (cumple1) & (suma==n4+n5+n6+n7):
                          for n8 in range(1,10):
                                                        cumple2 = (n8!=n1) and (n8!=n2) and (n8!=n3) and (n8!=n4)
                                                        cumple3 = (n8!=n5) and (n8!=n6) and (n8!=n7):
                            if cumple2 and cumple3:
                              for n9 in range(1,10):
                                cumple4=(n9!=n1) and (n9!=n2) and (n9!=n3) and (n9!=n4)
                                                                cumple5 = (n9!=n5) and (n9!=n6) and (n9!=n7) and (n9!=n8)
                                  if (cumple4) and (cumple5) and (suma==n7+n8+n9+n1):
                                    print("Secuencia: ", n1, " ", n2, " ", n3, " ", n4, " ", n5, " ", n6, " ", n7, " ", n8, " ", n9)
```

Figura 3.13. Código que implementa el programa del triángulo.

En primer lugar nos situamos sobre "*File*" en el menú y se selecciona el comando "*New*", como se muestra en la figura 3.14.

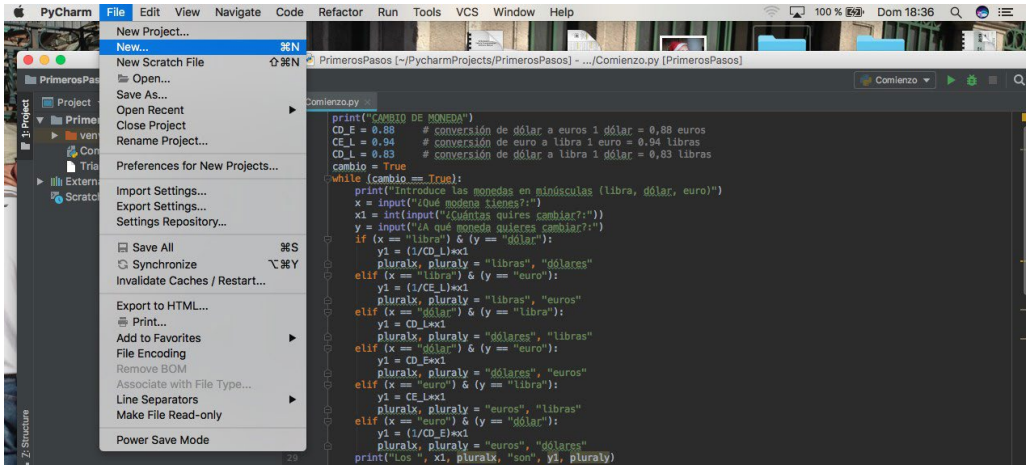


Figura 3.14. Pulsamos sobre File→New.

Nos aparece un menú como el mostrado en la figura 3.15, del que se selecciona "*Python File*".

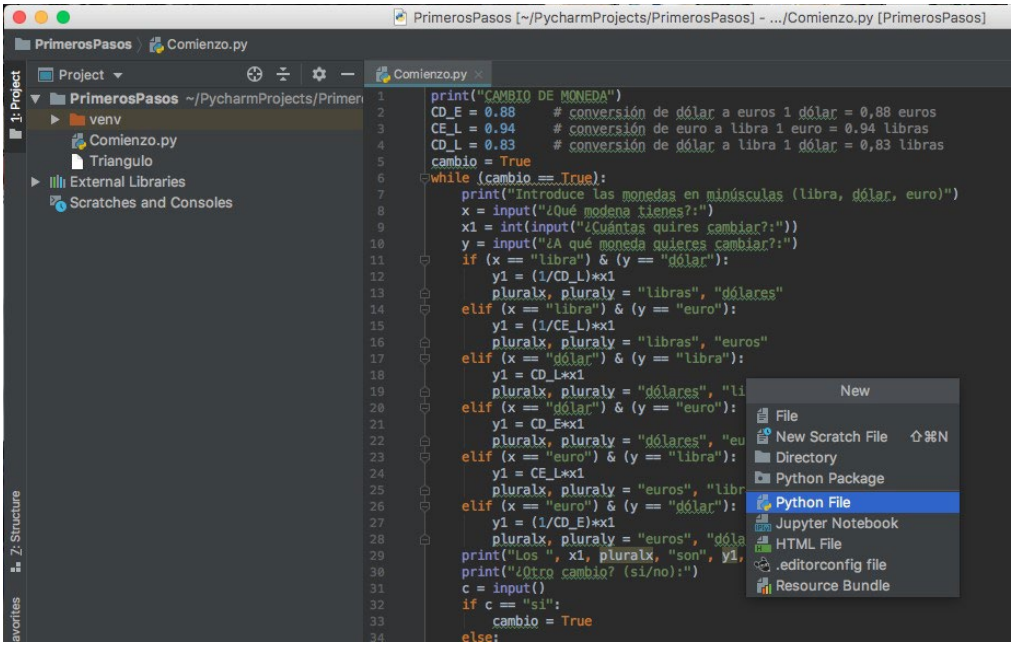


Figura 3.15. Seleccionamos Python File.

Ahora aparece un cuadro de diálogo para escribir el nombre del programa, como se muestra en la figura 3.16. Lo escribimos y pulsamos OK.

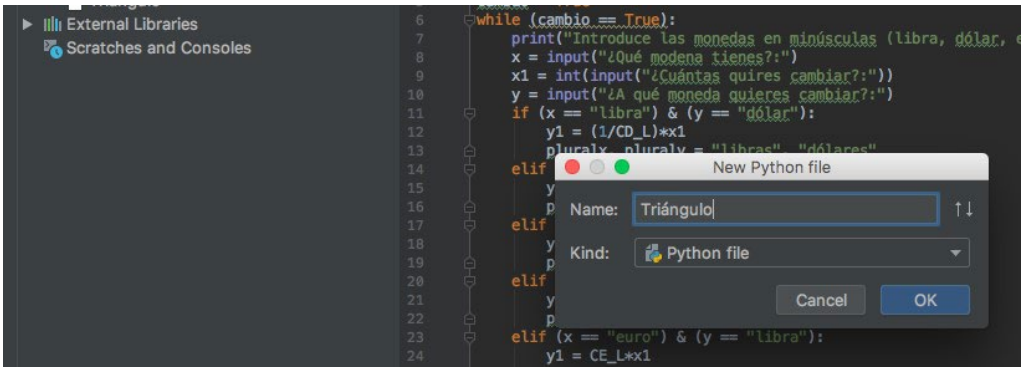


Figura 3.16. Escribimos el nombre del programa.

En la figura 3.17 vemos que tras pulsar OK, aparece el editor, donde se escribirán los programas, con el nombre del programa sobre una solapa con el nombre del programa activado. También aparece a su izquierda el nombre del programa anterior, ya que en realidad se verán los nombres de los programas vinculados a ese proyecto. El nombre de los programas también aparece en la columna de la izquierda en donde se muestra el conjunto de ficheros del proyecto.

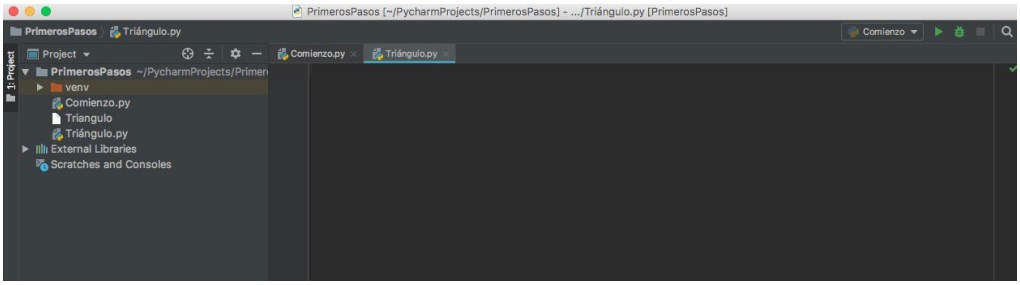


Figura 3.17. Estamos en el editor para escribir el código del programa.

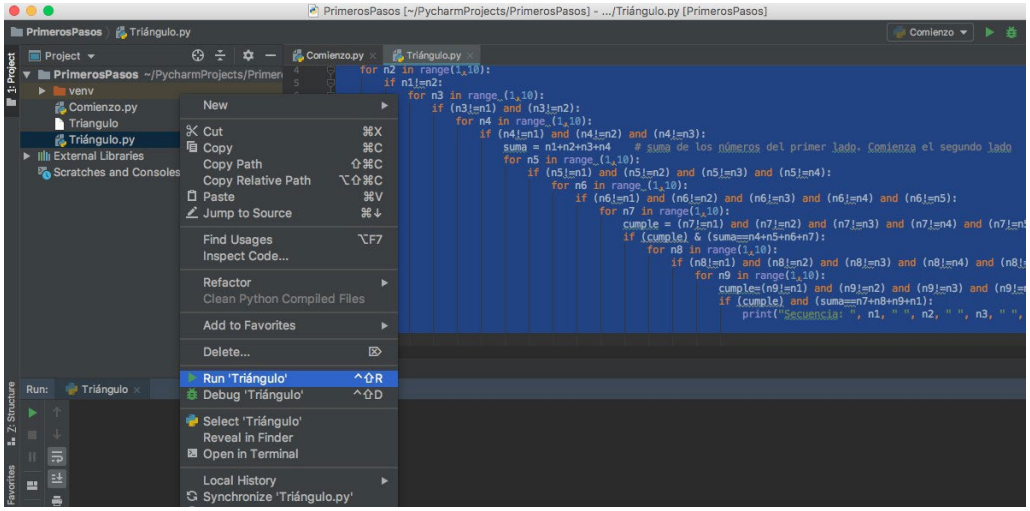


Figura 3.18. Ejecutamos el programa.

Para ejecutar el programa, dado que está activo el programa anterior (Comienzo), como se muestra en la parte superior derecha, hay que pulsar el botón derecho sobre Triángulo.py y de abre un desplegable como se muestra en la figura 3.18, del que se selecciona “Run ‘Triángulo’ ”. Tras ello, se ejecuta el programa y ya aparece en el desplegable superior derecha el nombre “Triángulo” del programa, como se muestra en la figura 3.19. Siempre que se ejecuta un programa, se ejecuta el nombrado en el desplegable de la parte superior derecha. Al pulsar ahora sobre ese desplegable, aparecen los nombres de los programas para seleccionar el que se desea ejecutar.

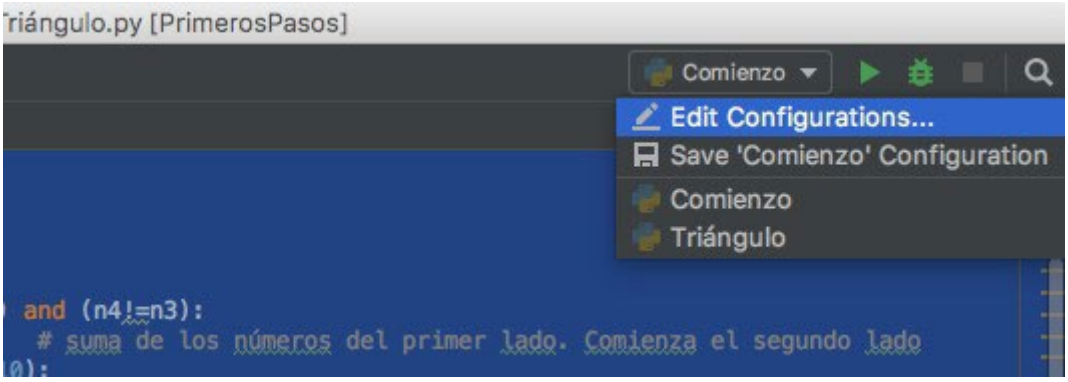


Figura 3.19. Se observa el nombre del programa.

Tras ejecutar el programa, resulta un conjunto grande de posibles combinaciones, algunas de las cuales son:

```
Secuencia:  1  2  9  7  3  5  4  6  8
Secuencia:  1  2  9  7  3  5  4  8  6
Secuencia:  1  2  9  7  5  3  4  6  8
Secuencia:  1  2  9  7  5  3  4  8  6
Secuencia:  1  3  7  9  2  4  5  6  8
Secuencia:  1  3  7  9  2  4  5  8  6
Secuencia:  1  3  7  9  4  2  5  6  8
Secuencia:  1  3  7  9  4  2  5  8  6
Secuencia:  1  3  8  7  2  6  4  5  9
Secuencia:  1  3  8  7  2  6  4  9  5
Secuencia:  1  3  8  7  6  2  4  5  9
Secuencia:  1  3  8  7  6  2  4  9  5
Secuencia:  1  4  9  3  5  7  2  6  8
Secuencia:  1  4  9  3  5  7  2  8  6
Secuencia:  1  4  9  3  7  5  2  6  8
Secuencia:  1  4  9  3  7  5  2  8  6
Secuencia:  1  5  9  2  4  8  3  6  7
Secuencia:  1  5  9  2  4  8  3  7  6
Secuencia:  1  5  9  2  8  4  3  6  7
Secuencia:  1  5  9  2  8  4  3  7  6
Secuencia:  1  5  9  4  2  6  7  3  8
Secuencia:  1  5  9  4  2  6  7  8  3
Secuencia:  1  5  9  4  6  2  7  3  8
```

```
Secuencia:  1   5   9   4   6   2   7   8   3
Secuencia:  1   6   7   3   4   8   2   5   9
Secuencia:  1   6   7   3   4   8   2   9   5
Secuencia:  1   6   7   3   8   4   2   5   9
Secuencia:  1   6   7   3   8   4   2   9   5
Secuencia:  1   6   8   2   5   7   3   4   9
Secuencia:  1   6   8   2   5   7   3   9   4
Secuencia:  1   6   8   2   7   5   3   4   9
Secuencia:  1   6   8   2   7   5   3   9   4
Secuencia:  1   6   8   4   3   5   7   2   9
Secuencia:  1   6   8   4   3   5   7   9   2
Secuencia:  1   6   8   4   5   3   7   2   9
Secuencia:  1   6   8   4   5   3   7   9   2
Secuencia:  1   6   8   5   2   4   9   3   7
Secuencia:  1   6   8   5   2   4   9   7   3
Secuencia:  1   6   8   5   4   2   9   3   7
Secuencia:  1   6   8   5   4   2   9   7   3
Secuencia:  1   7   3   9   2   4   5   6   8
Secuencia:  1   7   3   9   2   4   5   8   6
Secuencia:  1   7   3   9   4   2   5   6   8
Secuencia:  1   7   3   9   4   2   5   8   6
Secuencia:  1   7   6   3   4   8   2   5   9
Secuencia:  1   7   6   3   4   8   2   9   5
Secuencia:  1   7   6   3   8   4   2   5   9
Secuencia:  1   7   6   3   8   4   2   9   5
Secuencia:  1   8   3   7   2   6   4   5   9
Secuencia:  1   8   3   7   2   6   4   9   5
Secuencia:  1   8   3   7   6   2   4   5   9
Secuencia:  1   8   3   7   6   2   4   9   5
Secuencia:  1   8   6   2   5   7   3   4   9
Secuencia:  1   8   6   2   5   7   3   9   4
Secuencia:  1   8   6   2   7   5   3   4   9
Secuencia:  1   8   6   2   7   5   3   9   4
Secuencia:  1   8   6   4   3   5   7   2   9
Secuencia:  1   8   6   4   3   5   7   9   2
```

```
Secuencia:  1   8   6   4   5   3   7   2   9
Secuencia:  1   8   6   4   5   3   7   9   2
Secuencia:  1   8   6   5   2   4   9   3   7
Secuencia:  1   8   6   5   2   4   9   7   3
Secuencia:  1   8   6   5   4   2   9   3   7
Secuencia:  1   8   6   5   4   2   9   7   3
Secuencia:  1   9   2   7   3   5   4   6   8
Secuencia:  1   9   2   7   3   5   4   8   6
Secuencia:  1   9   2   7   5   3   4   6   8
Secuencia:  1   9   2   7   5   3   4   8   6
Secuencia:  1   9   4   3   5   7   2   6   8
Secuencia:  1   9   4   3   5   7   2   8   6
Secuencia:  1   9   4   3   7   5   2   6   8
Secuencia:  1   9   4   3   7   5   2   8   6
Secuencia:  1   9   5   2   4   8   3   6   7
Secuencia:  1   9   5   2   4   8   3   7   6
Secuencia:  1   9   5   2   8   4   3   6   7
Secuencia:  1   9   5   2   8   4   3   7   6
Secuencia:  1   9   5   4   2   6   7   3   8
Secuencia:  1   9   5   4   2   6   7   8   3
Secuencia:  1   9   5   4   6   2   7   3   8
Secuencia:  1   9   5   4   6   2   7   8   3
Secuencia:  2   1   9   7   4   5   3   6   8
Secuencia:  2   1   9   7   4   5   3   8   6
Secuencia:  2   1   9   7   5   4   3   6   8
Secuencia:  2   1   9   7   5   4   3   8   6
Secuencia:  2   1   9   8   3   4   5   6   7
Secuencia:  2   1   9   8   3   4   5   7   6
Secuencia:  2   1   9   8   4   3   5   6   7
Secuencia:  2   1   9   8   4   3   5   7   6
Secuencia:  2   3   7   8   1   6   5   4   9
Secuencia:  2   3   7   8   1   6   5   9   4
```

RESUMEN

- Las estructuras de control son: selecciones, que permiten ejecutar una u otra parte de un programa en función de que se cumpla o no una determinada condición e Iteraciones que repiten la ejecución de un bloque de programa en función de que se cumpla una condición.

- Las selecciones tienen la forma mostrada en la tabla que sigue:

`if B1:` `  S1`	Si se cumple la condición B1 (expression lógica con valor True) se ejecuta la secuencia S1, en caso contrario (valor False), no hace nada.

`if B1:`
`  S1`
`else:`
`  S2`
	Si se cumple la condición B1 se ejecuta la secuencia S1, en caso contrario se ejecuta S2.

`if B1:`
`  S1`
`elif B2:`
`  S2`
`else:`
`  S3`
	Si (if) se cumple la condición B1 se ejecuta la secuencia S1. Si no, si (elif) se cumple B2 se ejecuta S2. Si no se cumple ninguna (else), se ejecuta S3.
	elif es una contracción de else if (si no, si).

- La composición iterativa tiene la forma:

```
while B:
        S
```

  al principio de la ejecución, se comprueba la condición B y si toma valor True (verdad) se ejecuta la instrucción S, usualmente conocida como cuerpo de la iteración; esto se repite hasta que la condición B no se cumple. Caso de que de entrada no se cumpla la condición, el bloque S no se ejecuta.

- Otra composición iterativa es la iteración *for* que en Python itera sobre los elementos de una secuencia (lista, rango o cadena de caracteres) en el orden en que aparecen en la secuencia. Se usa en la forma: `for s in sec:`

- Se puede modificar el comportamiento de una iteración utilizando una selección if para detenerla si se cumple una cierta condición. La sentencia *break* hace que termine la iteración.

- Si en lugar de break se usa la sentencia *continue* simplemente se salta un paso de la iteración.

- También puede operar sobre un rango de valores enteros en la forma: `for x in range(m, n)`.

- La iteración *for* puede contener una sentencia *else* que se ejecuta cuando termina la iteración.
- Python no restringe la visibilidad de una variable al interior del bloque en el que fue asignada.