Estructuras Secuenciales y Estructurados básicos

Índice

1. Resolución de problemas simples

En un algoritmo intervienen tres elementos: los datos, los procesos y los resultados. Dado un problema, para plantear el algoritmo que permita resolverlo es conveniente entender correctamente el enunciado y tratar de deducir y caracterizar a partir del mismo los elementos ya indicados:

a. Comenzar identificando los resultados pedidos, quedando claros los objetivos.
b. Individualizar los datos con que se cuenta para determinar si son suficientes para llegar a los resultados pedidos.
c. Si los datos son completos y los objetivos claros, se intentan plantear los procesos necesarios para convertir esos datos en resultados esperados.

Para resolver un problema, con los conceptos aprendidos, será necesario dividir el problema en 3 fases:

i. Identificación de componentes: Para cada componente identificado, asociar un nombre (usado como variable).

ii. Plantear el algoritmo: Definir los pasos necesarios para obtener el resultado esperado de los datos y procesos.

iii. Desarrollo del programa: Se escribe en lenguaje de Python el código correspondiente a los pasos definidos en la planeación del algoritmo.

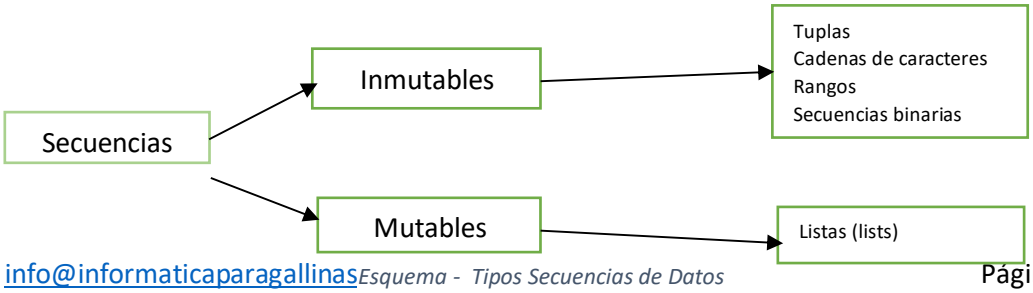
2. Secuencias de datos

Aquellos tipos de datos que solo admiten que una variable pueda contener un único valor de ese tipo se llaman tipos simples. Los tipos int, float y bool, son ejemplos de tipos simples. Una variable de este tipo puede representarse como un recipiente con un único compartimento para almacenar un único elemento.

En Python, existen tipos de datos que permiten que una misma variable pueda contener varios valores al mismo tiempo, conocidos como tipos compuestos o tipos estructurados, y por esa razón una variable de tipo estructurado se suele designar también como una estructura de datos.

Hay 2 tipos de secuencias de datos en Python:

i. Secuencia de datos inmutable: Simplemente un conjunto finito de datos cuyos valores y orden de aparición no pueden modificarse una vez asignados. Las cadenas de caracteres son un ejemplo de secuencias inmutables.
ii. Secuencia de datos mutables: Aquellas secuencias en las que los valores individuales pueden modificarse luego de ser asignados.



Esquema - Tipos Secuencias de Datos

Ejercicio 1: Ordenar las siguientes secuencias de datos entre mutables e inmutables:

- **Tuplas**
- **Secuencias binarias**
- **Listas**
- **Cadenas de caracteres**
- **Rangos**

Una cadena de caracteres puede entenderse como un contenedor dividido en tantas casillas como caracteres tenga esa cadena. Si el programador necesita almacenar varios números en el mismo contenedor, puede usar las tuplas (entre otras alternativas) en Python.

Una *tupla t* es una secuencia inmutable de datos que pueden ser de tipos diferentes. Hay varias formas de definir una tupla que contenga ninguno, uno o varios valores iniciales:

1. Usando un par de paréntesis vacíos para denotar una tupla vacía.
2. Usando una coma al final, para denotar una tupla de un solo valor.
3. Separando los ítems por comas → t = a, b, c
4. Usando la función *tuple()* → ***t = tuple('<cadena>')*** → donde la función crea una tupla *t* por cada uno de los caracteres de la <cadena>

Cada casillero de una secuencia de cualquier tipo está asociado a un numero de orden llamado índice, mediante el cual se puede acceder a ese casillero en forma individual. La secuencia de índices que identifica a los casilleros de una secuencia de datos comienza desde cero hasta llegar al número de caracteres menos 1 → ***R=[0-(n-1)]***

Para acceder a un elemento individual de una secuencia, basta con escribir el identificador de la misma y luego agregar entre corchetes el índice del casillero a acceder:

nom = 'sergio' → *t[0] = 's'; t[1] = 'e'; t[2] = 'r'; t[3] = 'g'; t[4] = 'i'; t[5] = 'o'*

Cuando se trabaja con secuencias de tipo inmutable, cualquier intento de modificar el valor de un casillero individual accediéndolo por su índice producirá un error. La modificación de un elemento de una secuencia está permitida siempre y cuando esa secuencia sea del tipo *mutable*, como en el caso de las *listas*.

Es perfectamente válido que una tupla contenga elementos de tipos diferentes, los cuales, si se desea procesar por separado cada elemento, se usan los índices como se indicó (última función resaltada en verde arriba) y se pueden introducir variables temporales a criterio y necesidad del programador.

En Python, una expresión de asignación puede estar formada por tuplas en ambos lados de la expresión: una secuencia de variables separada por comas en el lado izquierdo, y otra secuencia con la misma cantidad de valores, variables y|o expresiones del lado derecho separadas por comas: ***x, y = a*3, a**2***

Se pueden realizar intercambios de variables en Python recurriendo a la asignación de tuplas en forma directa. Esto quiere decir que sin tener que recurrir a una variable auxiliar, se puede realizar el intercambio de variables:

Para ***a = 5 y b = 8*** → ***a, b = b, a*** (A =8 y b= 5)

Ejercicio 2: Crear un fichero llamado cambNum.py en el cual se intercambie el valor correspondiente a las variables *A* y *B*, siendo estos inicialmente 10 y 13 respectivamente. Mostrar por pantalla sus valores iniciales y luego de hacer el cambio de valores mostrar nuevamente el valor para *A* y *B* actualmente.

Ejercicio 3: Crear un fichero llamado cambLet.py en el cual esté predefinida la variable n = 'sol' y hacer que muestre por pantalla la cadena a la inversa mostrándolo por pantalla. El mensaje debe ser (sol al revés es '<solución>'). Usar los índices de tuplas para la ordenación a la inversa de la variable.

3. Cadenas de caracteres: Caracteres especiales

Un *string* se define como una secuencia inmutable de caracteres y por eso, le son aplicables todos los operadores, funciones y métodos propios del manejo de secuencias.

Un literal (una constante) de tipo *string* es una secuencia de caracteres delimitada por comillas dobles o por comillas simples: ***nombre = "Pedro"***

Si el delimitador usado fue una comilla doble, entonces la comilla simple puede usarse dentro de la cadena como carácter directo y viceversa: ***apellido = "O'Dogerthy"***

- ***Caracteres Especiales:***

  Un carácter de escape se forma con un símbolo de barra (\) seguido de algún carácter especial que actúa en forma de código, interpretando la secuencia **\<carácter>** como un solo y único código. En algunos casos, el carácter de escape provoca algún efecto en la salida por consola:

  - **\n** → Produce el efecto de un salto de línea en la salida por consola cuando aparece dentro de una cadena. ***print('Hola\nMundo')***
  - **\t** → Provoca un espaciado en la consola de salida, equivalente a una "sangría" (tabulación). ***print('Hola\tMundo')***
  - **\'** y **\"** → Representan literalmente comillas simples o dobles y permiten entonces su uso dentro de cadenas en forma directa, sin importar si estas cadenas fueron abiertas con el mismo delimitador.
    - ***apellido = 'O\'Doguerthy'***
    - ***aka = "Sergio \"la barba\" Romero"***
  - Índice: Puede usarse para acceder a esos caracteres de una cadena. Lo que no puede hacer es cambiar el calor de una cadena ya creada y almacenada en memoria. Pero no hay ningún problema en que una variable que apuntaba a una cadena pase a apuntar a otra diferente.

- Suma (+) → Permite la concatenación o unión de dos o más cadenas, creando una nueva cadena que contiene las cadenas originales en el orden en que se unieron.
  - *nombre = 'Sergio'*
  - *apellido = "Pérez"*
  - *completo = nombre + " " + apellido*
- Producto (*) → Permite repetir una cadena varias veces concatenando el resultado. *replicado = nombre * 4*

Ejercicio 4: Crear un fichero llamado carEsp.py en el cual se pida introducir el nombre y apellido por teclado para luego mostrar por pantalla un ejemplo de cada carácter especial explicado en este documento. Nota: los caracteres especiales son: el de salto de línea, el de tabular, las comillas dobles y simples, la concatenación de nombre y apellido, la repetición de la concatenación 3 veces separadas por saltos de línea y el cambio del contenido de la variable nombre por otro nuevo introducido por teclado.

4. Funciones básicas

Python provee de un amplio conjunto de funciones internas listas para usar. Estas funciones se instalan en la instalación del programa de Python. Algunas de las funciones más usadas en Python son:

Función	Descripción	Ejemplo
abs(x)	Retorna el valor absoluto de X	a = abs(x)
bin(x)	Retorna en binario el valor X	a = bin(x)
chr(i)	Retorna el ASCII del número i	a = chr(i)
divmod(a,b)	Retorna cociente y resto de a // b	a = divmod(a,b)
float(x)	Convierte a coma flotante la cadena X	a = float(x)
hex(x)	Retorna en hexadecimal el valor X	a = hex(x)
input(x)	Obtiene una cadena por teclado	p = input('x')
int(x)	Convierte a número entero la cadena X	a = int(x)
len(s)	Retorna la longitud (cantidad elementos) del objeto S	n = len(s)
max(a,b,*args)	Retorna el mayor de los parámetros	a = max(a,b,c,d)
min(a,b,*args)	Retorna el menor de los parámetros	a = min(a,b,c,d)
oct(x)	Retorna el octal del valor X	a = oct(x)
ord(c)	Retorna el entero del valor ASCII	i = ord(c)
pow(x,y)	Retorna el valor de X elevado a Y	a = pow(x,y)
print(p)	Muestra el valor p por pantalla	a = print(p)
round(x,n)	Retorna X redondeado a N decimales	a = round(x, 2)
str(x)	Retorna una string del objeto X	s = str(x)

Tabla - Funciones comunes en Python

5. Ejercicios propuestos

Son los mismos ejercicios que se han propuesto anteriormente, pero recopilados al final del tema por si se prefiere poner todo en práctica una vez se haya terminado la lectura del tema.

Ejercicio 1: Ordenar las siguientes secuencias de datos entre mutables e inmutables:

- **Tuplas**
- **Secuencias binarias**
- **Listas**
- **Cadenas de caracteres**
- **Rangos**

Ejercicio 2: Crear un fichero llamado cambNum.py en el cual se intercambie el valor correspondiente a las variables *A* y *B*, siendo estos inicialmente 10 y 13 respectivamente. Mostrar por pantalla sus valores iniciales y luego de hacer el cambio de valores mostrar nuevamente el valor para *A* y *B* actualmente.

Ejercicio 3: Crear un fichero llamado cambLet.py en el cual esté predefinida la variable n = 'sol' y hacer que muestre por pantalla la cadena a la inversa mostrándolo por pantalla. El mensaje debe ser (sol al revés es '<solución>'). Usar los índices de tuplas para la ordenación a la inversa de la variable.

Ejercicio 4: Crear un fichero llamado carEsp.py en el cual se pida introducir el nombre y apellido por teclado para luego mostrar por pantalla un ejemplo de cada carácter especial explicado en este documento. Nota: los caracteres especiales son: el de salto de línea, el de tabular, las comillas dobles y simples, la concatenación de nombre y apellido, la repetición de la concatenación 3 veces separadas por saltos de línea y el cambio del contenido de la variable nombre por otro nuevo introducido por teclado.

Ejercicio 5: Crear un fichero llamado funBas.py en el cual se pida introducir un numero en coma flotante por teclado y otro de tipo entero, mostrar por pantalla el valor absoluto del número en coma flotante, potenciar el mismo al número entero, mostrar el carácter ASCII del número entero.

Ejercicio 6: Crear un fichero llamado maxNum.py en el cual se pida introducir por teclado 5 números y saque por pantalla cual es el número mayor introducido.

Ejercicio 7: Crear un fichero llamado minNum.py en el cual se pida introducir por teclado 5 números y saque por pantalla cual es el número menor introducido.

Ejercicio 8: Crear un fichero llamado cocRes.py en el cual se pida introducir 2 números por teclado de tipo entero y se muestre por pantalla el cociente y el resto correspondientes a esa división entera.

Ejercicio 9: Crear un fichero llamado notaMedia.py en el cual se pida introducir las notas correspondientes a las 3 evaluaciones de una asignatura (con 2 decimales máximo), y se muestre por pantalla la nota final redondeada a 1 decimal.

Ejercicio 10: Crear un fichero llamado nom_3let.py en el cual se pida introducir por teclado el nombre, se debe de contar el número de caracteres que contiene el nombre y mostrar por pantalla el mensaje (<nombre> tiene <numero> caracteres.). También, debe imprimir el valor individual correspondiente a las 3 primeras letras del nombre ordenándolas con saltos de línea entre cada letra.

Ejercicio 11: Crear un fichero llamado cambioBase1.py que pida la introducción de un número en decimal por teclado y devuelva por pantalla su cambio de base a binario, octal, hexadecimal y su valor en ASCII de dicho número. A la hora de imprimir por pantalla se deberá ver el numero introducido seguido de una descripción de la acción ejecutada y el resultado correspondiente a dicha acción. (Ej: 65 en binario es 0b1000001). Usar las funciones vistas *bin(), oct(), hex(), chr().*

Ejercicio 12: Crear un fichero llamado cambioBase2.py que pida la introducción de un carácter por teclado y devuelva por pantalla su cambio de base a binario, octal, decimal y hexadecimal correspondiente al valor de dicho carácter. A la hora de imprimir por pantalla se deberá ver el carácter introducido seguido de una descripción de la acción ejecutada y el resultado correspondiente a dicha acción. (Ej: A en binario es 0b1000001). Usar las funciones vistas *bin(), oct(), ord(), hex().*

Soluciones a los ejercicios propuestos → (**https://mega.nz/#!giw0AZoL!cRQ2UDjStivaGuA9WxM_UrClIFiZPMx4jjQtVbVg4g4**)