INFORMATORIO- UNIDAD II: INTRODUCCIÓN A LA PROGRAMACIÓN

Apunte de Teoría

Elaborado por Noelia Pinto – Blas Cabas Geat

Tabla de contenido

Unidad II: Introducción a la Programación

Contenido Temático

a. Datos. Clasificación de los datos. Tipo de datos. Ejemplos en Python.
b. Constantes y Variables. Estados. Definición en Python.
c. Acciones. Clasificación de las acciones.
d. Operadores, operandos y expresiones. Prioridad de Operadores y evaluación de expresiones.
e. Estructura de un Algoritmo. Estructura de programa en Python.
f. Conceptos Iniciales de Testing.

Datos. Clasificación de los datos. Tipo de datos

Concepto de Dato

Son todos aquellos elementos considerados como unidades de tratamiento dentro de un sistema de proceso de datos. Un **dato** es toda aquella información característica de una entidad, que es susceptible de tratamiento en un programa informático.

Los datos pueden ser básicamente de dos tipos, denominados **datos de entrada** que son aquellos pendientes de proceso o elaborar y **datos de salida** que son aquellos obtenidos tras el proceso de los datos de entrada. Al conjunto de los datos que posee significado se le denomina **información**.

Por ello, para su correcto tratamiento, todos los datos que forman dicha información deben estar clasificados de tal forma que los programas sean capaces de realizar operaciones sobre ellos sin dar lugar a error o fallos por parte del sistema.

> ***Ejemplo***
> *Situación: Ubicar la oficina del Informatorio*
> *Dato: Edificio de Bolsa de Comercio*
> *Información: Frondizi 174 – 4to Piso*

El dato no tiene valor semántico (sentido) en sí mismo, pero convenientemente tratado o procesado se puede utilizar en la realización de cálculos o toma de decisiones.

Clasificación

El **tipo de un dato** es el conjunto de valores que puede tomar durante el programa. Si se le intenta dar un valor fuera del conjunto se producirá un error.

La asignación de tipos a los datos tiene dos objetivos principales:

- Por un lado, detectar errores en las operaciones
- Por el otro, determinar cómo ejecutar estas operaciones

Para el diseño de un programa es importante establecer cuáles son las estructuras de datos que se van a utilizar, con el objeto de establecer las operaciones que sobre dichos datos se pueden realizar, para lo cual debemos proporcionar información al sistema acerca de los mismos. Los

datos manejados en un programa deben llevar asociado por tanto un identificador, un tipo y un valor.

Pueden realizarse distintas clasificaciones de los datos, por ejemplo, dependiendo de:

a) si varían o no durante la ejecución de un programa, pueden ser variables o constantes.

b) su naturaleza (lo que representan), tamaño que ocupan en memoria, formato de codificación y funcionalidad (qué operaciones se pueden realizar con ellos), pueden clasificarse en numéricos, texto, booleanos, etc.

c) cómo y en qué lugar de la memoria se almacenan, serán estáticos o dinámicos.

Clasificación de acuerdo a cómo se almacenan en memoria

En los datos **estáticos** su tamaño y forma es constante durante la ejecución de un programa y por tanto se determinan en tiempo de compilación. El ejemplo típico son los arrays. Tienen el problema que hay que dimensionarlos previamente, que puede causar desperdicio o falta de memoria.

En los datos **dinámicos** su tamaño y forma es variable (o puede serlo) a lo largo de un programa, por lo que se crean y destruyen en tiempo de ejecución. Esto permite dimensionar la estructura de datos de una forma precisa: se va asignando memoria en tiempo de ejecución según se va necesitando. Las variables de tipo puntero son las que nos permiten referenciar datos dinámicos.

Tipos de Datos

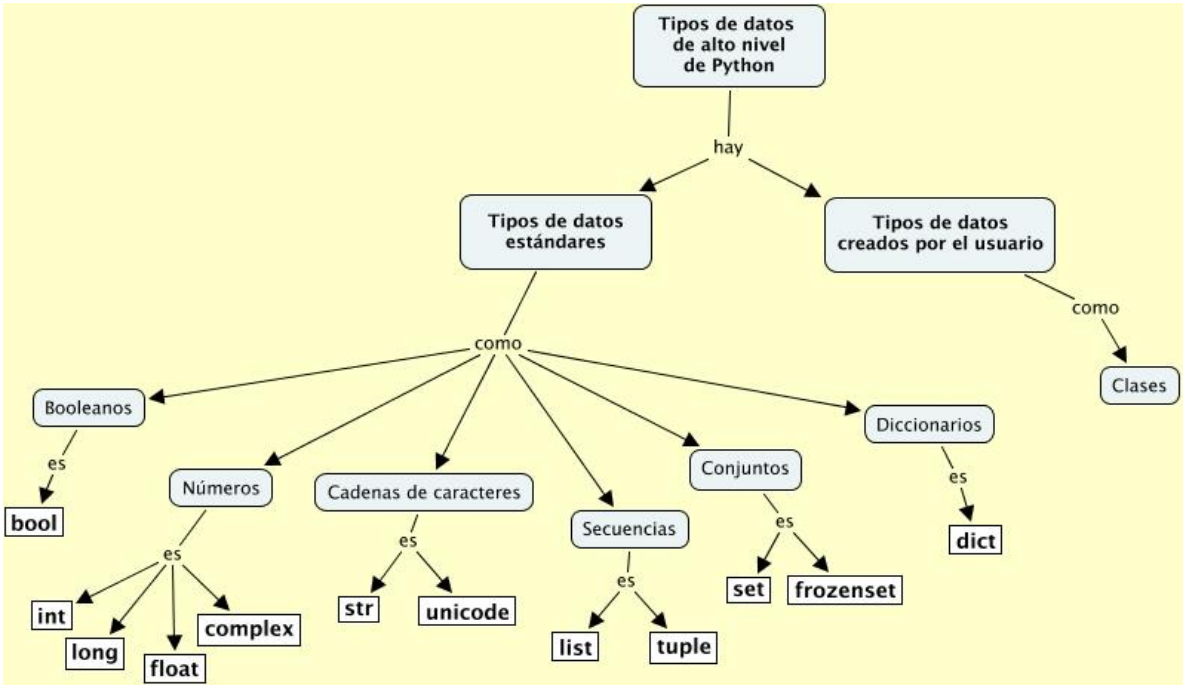

Figura 1 – Clasificación de los Tipos de Datos en Python

Ejemplos:

DATOS SIMPLES			
Numéricos		Alfanuméricos	Lógicos
Se utilizan para contar, acumular valores, almacenar el resultado de una operación matemática, etc.		Se utilizan para almacenar nombres, texto, mensajes de error, mensajes de inicio, etc.	Se usan para almacenar información binaria o resultado de operaciones lógicas. Es decir, por ejemplo, se almacena el resultado de verificar si un número es o no par. Solo tienen dos valores posibles.
Enteros	Reales	C = 'Ingrese usuario y contraseña'	D = Verdadero
A = 2 + 3 A = 5	B = 2.5 * 3 B = 7.5		

Tipos de datos: Números

1) **Enteros:**

En Python se pueden representar mediante el tipo `int` (de integer, entero) o el tipo `long` (largo). La única diferencia es que el tipo `long` permite almacenar números más grandes. Es aconsejable no utilizar el tipo `long` a menos que sea necesario, para no malgastar memoria.

Al asignar un número a una variable esta pasará a tener tipo `int`, a menos que el número sea tan grande como para requerir el uso del tipo `long`.

```
# type(entero) devolvería int
entero = 23
```

También podemos indicar a Python que un número se almacene usando `long` añadiendo una L al final:

```
# type(entero) devolvería long
entero = 23L
```

El literal que se asigna a la variable también se puede expresar como un octal, anteponiendo un cero:

```
# 027 octal = 23 en base 10
entero = 027
```

O bien en hexadecimal, anteponiendo un 0x:

```
# 0×17 hexadecimal = 23 en base 10
entero = 0×17
```

2) Reales:

En Python se expresan mediante el tipo `float`. Desde la versión de Python 2.4 contamos también con un nuevo tipo `decimal`, para el caso de que se necesite representar fracciones de forma más precisa. Sin embargo este tipo está fuera del alcance de este curso, y sólo es necesario para el ámbito de la programación científica y otros relacionados. Para aplicaciones normales se puede utilizar el tipo `float`, aunque teniendo en cuenta que los números en coma flotante no son precisos (ni en este ni en otros lenguajes de programación).

Para representar un número real en Python se escribe primero la parte entera, seguido de un punto y por último la parte decimal.

```
real = 0.2703
```

Tipos de datos: Cadenas

Las cadenas no son más que texto encerrado entre comillas simples (‘cadena’) o dobles (“cadena”). Dentro de las comillas se pueden añadir caracteres especiales de escape con \, como \n, el carácter de nueva línea, o \t, el de tabulación.

Tipos de datos: Booleanos

Estos valores son especialmente importantes para las expresiones condicionales y los bucles, como veremos más adelante. En Python se definen usando el tipo `bool`.

Constantes y Variables. Estados

Conceptos

Las ***constantes*** son datos cuyo valor no puede ser cambiado en el transcurso de un programa. El ejemplo clásico para representar una constante es: el número PI, representando al valor 3.1416.

Las ***variables*** son datos que pueden sufrir cambios en su valor durante la ejecución de un programa, o un algoritmo. Por ejemplo, la velocidad de un auto, el promedio de edad de los habitantes de una ciudad, etc.

Declaración de variables

Para utilizar las variables en un programa hay que declararla.

Primero se declaran las variables y luego se inicia las lecturas o procesos de la misma.

Para declarar una variable necesitamos definir el *identificador* y su tipo.

Se llama *identificador*, al nombre de la variable y la forma en que nos vamos a referir a ella en el programa. Se deben evitar usar espacios en blanco, símbolos especiales, palabras reservadas, etc.

Al declarar una variable hay que definir su *tipo*, y solo almacenará datos del tipo especificado. Luego la variable debe inicializarse[1] con algún valor. En algunos lenguajes no es necesario definir el tipo, pues éste queda implícito al comenza a usar la variable en el programa.

Clasificación de las Variables

a) Por su contenido

- *Variables Numéricas*: Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal.

  Ejemplo:

  iva = 0.15 | pi = 3.1416 | costo = 2500

[1] Cuando hacemos referencia a la acción de Inicialización de variable, estamos indicando que se debe dar un valor inicial a la misma. Puede ocurrir que no sea necesario la inicialización en algunos casos.

- *Variables Lógicas*: Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.
- *Variables Alfanuméricas*: Almacenan caracteres alfanuméricos (letras, números y caracteres especiales).

  Ejemplo:

  letra = 'a' | apellido = 'lopez' | direccion = 'Av. Libertad #190'

b) Por su uso

- *Variables de Trabajo*: Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa.

  Ejemplo:

  suma = a + b /c
- *Contadores*: Se utilizan para llevar el control del número de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno. Por ejemplo, contar la cantidad de alumnos mayores de 21 años.
- *Acumuladores*: Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente. Por ejemplo, sumar importes de compra

En Python ni existen constantes ni existen variables, tan sólo objetos con los que hacer modificaciones. Lo que mal-llamamos variables (o constantes) no son otra cosa que "referencias" a objetos, como etiquetas para poder identificarlos.

Estado de las variables

Se denomina **estado** de una variable, al valor que contiene en un determinado instante de tiempo Ti.

Los estados más significativos de las variables son:

- *Estado Inicial* (T_0): El estado inicial de una variable, ocurre luego de ser declarada, en el instante 0 de ejecución, y es siempre indefinido.
- *Estados Intermedios* (T_1, T_2,…, T_m): Los estados intermedios ocurren a partir del instante de tiempo T_1. Y van registrando la actualización de valores que afectan a la variable. Los estados intermedios pueden ser 1 o más.
- *Estado Final* (T_f): El estado final de una variable registra el último cambio de estado que sufre la misma, y representa el resultado final. No necesariamente los valores de la variable en el instante T_i y en el instante T_f serán diferentes, depende de las operaciones que se realicen.

Ejemplo 1
Describa la tabla de estados de la variable X, al realizarse las siguientes operaciones:

*a) X = 2 * 25*
b) X = X + 4 - 10
c) X = X / 2

Solución

	Operación	Resultado
$T_0 = E_i$	?	?
T_1	X = 2*25	50
T_2	X = X+4-10	44
$T_3 = E_f$	X = X / 2	22

El estado final de la variable X es → E_f = 50

Acciones. Clasificación de las acciones

Conceptos

Una **acción** es un acontecimiento producido por un actor, que tiene lugar durante un período de tiempo finito y produce un resultado bien determinado.

Clasificación de las Acciones

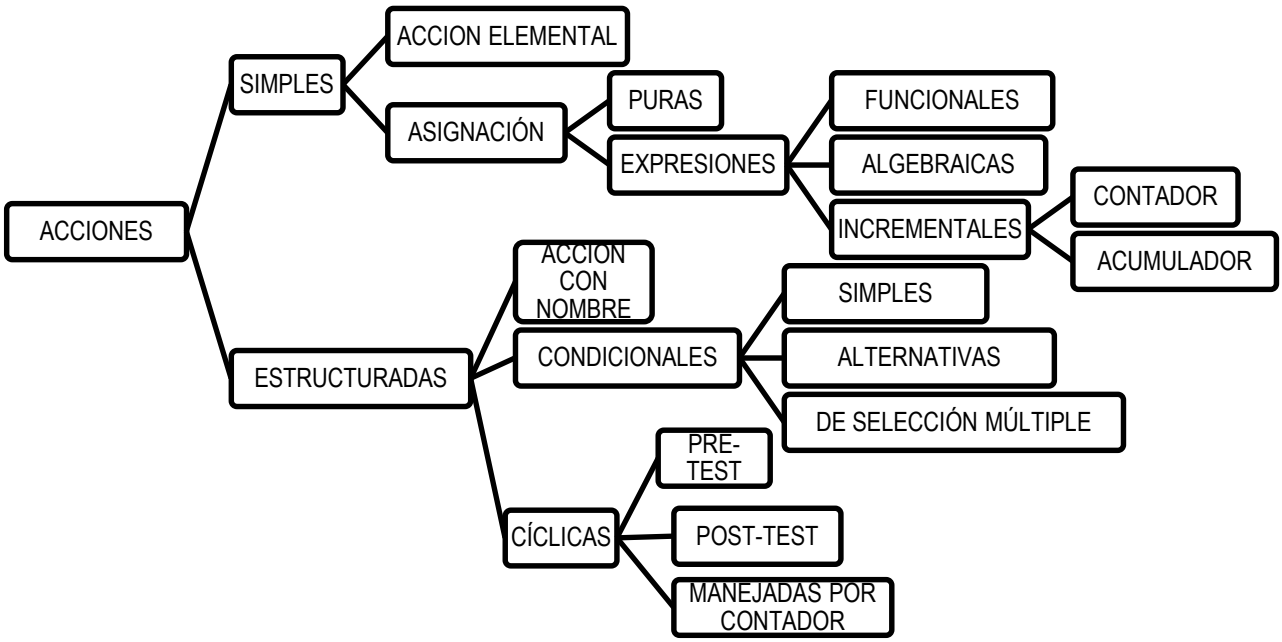

Figura 2 – Clasificación de las Acciones

Acciones Simples[2]

a) *Acciones Elementales*: El procesamiento de la información requiere de la entrada de datos, que luego serán transformados por una serie de operaciones en el resultado o datos de salida.

   Las operaciones de entrada permiten leer determinados valores y asignarlos a variables (asignación interna), esta operación es la acción elemental de Lectura y en Python se representa mediante **input()**.

   Las operaciones de salida permiten mostrar resultados en pantalla, imprimirlos, listarlos, etc. Esta es la acción elemental de Escritura y en Python se simboliza usando **print()**.

b) *Asignación*: Es la acción que da valor a una variable (asignación externa). Operación que le transfiere contenido a la variable. En Python el símbolo de asignación es **= .**

   Se dice que es una Asignación Pura si transfiere un valor a la variable. Por ejemplo, X=5.

   Por el contrario es una Asignación de Expresión, cuando a la variable se la asocia con una expresión que puede ser una función, una expresión algebraica o una operación incremental (con incrementos constantes o no). Por ejemplo,

Expresión Funcional	**Expresión Algebraica**	**Expresión Incremental**
X=EsPar(2)	X=(2 * X)+16	X=X+1

[2] Más adelante se profundiza en las Acciones Estructuradas.

Operadores, operandos y expresiones. Prioridad de los operadores. Evaluación de las expresiones

Operadores

Los operadores nos permiten manipular datos, sean variables, constantes, otras expresiones, objetos, atributos de objetos, entre otros, de manera que podamos:

a) transformarlos,

b) usarlos en decisiones para controlar el flujo de ejecución de un programa

c) formar valores para asignarlos a otros datos

El tipo de datos involucrado en una expresión se relaciona con los operadores utilizados.

Operandos

Son los datos a los que se les asocia un operador para efectuar una determinada operación con sus valores. Recordemos que los datos pueden ser variables o constantes.

Tipos de Operadores

Los operadores que veremos en nuestro curso son:

1- *Operadores Aritméticos*: Sus operandos son matemáticos.

Operador	**Descripción**
+	Suma
-	Resta
*	Multiplicación
/	División real
^	Potencia
%	Resto de la División
//	Cociente de la División
abs(x)	Valor absoluto
round(x [,n])	Redondeo del valor. X=ROUND(5.5) → X = 6 X=ROUND(5.3) → X = 5
cmp(x,y)	-1 if x < y, 0 if x == y, or 1 if x > y
max(x1,x2,…xn) min(x1,x2,…xn)	x = 5.5 y = 2.4 z = 3 print(max(x,y,z))→5.5

2- *Operadores Relacionales*. Sus operandos pueden ser numéricos o alfanuméricos.

Operador	**Descripción**	**Uso**
<	Menor a	X < 2
<=	Menor o igual a	X <= 2
>	Mayor a	X > 2
>=	Mayor o igual a	X >= 2
==	Igual a	X == 2
!=	Distinto a	X != 2

3- *Operadores Lógicos*: Sus operandos son expresiones relacionales.

Operador	**Descripción**	**Uso**
AND	Operador Lógico AND	x > 0 and x < 10
OR	Operador Lógico OR	n % 2 == 0 or n % 3 == 0
NOT	Operador Lógico NOT	not(x > y)

Tabla de valores

A Y B		
A	**B**	**RESULTADO**
V	V	V
V	F	F
F	V	F
F	F	F

A O B		
A	**B**	**RESULTADO**
V	V	V
V	F	V
F	V	V
F	F	F

Prioridad de los Operadores. Evaluación de las Expresiones

Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero.

Dentro de una misma expresión los operadores se evalúan en el siguiente orden:

1. ^
2. *, /, %, //, ROUND, ABS
3. +, -

Entonces para saber cómo es evaluada una expresión, debemos conocer 3 cuestiones:

- la prioridad de ejecución de los operadores involucrados
- la evaluación se realiza de izquierda a derecha
- el uso de paréntesis permite la alteración de la prioridad de ejecución.

Python: Características. Estructura de un Programa en Python

Introducción a Python

Dentro de los **lenguajes informáticos**, Python, pertenece al grupo de los **lenguajes de programación** y puede ser clasificado como un **lenguaje interpretado, de alto nivel, multiplataforma**, de **tipado dinámico y multiparadigma**. A diferencia de la mayoría de los lenguajes de programación, **Python nos provee de reglas de estilos**, a fin de poder escribir código fuente más legible y de manera estandarizada. Estas reglas de estilo, son definidas a través de la ***Python Enhancement Proposal* Nº 8**[3]**.**

Lenguaje Interpretado

Es el lenguaje cuyo código *no necesita ser preprocesado mediante un compilador*, eso significa que la computadora es capaz de ejecutar la sucesión de instrucciones dadas por el programador sin necesidad de leer y traducir exhaustivamente todo el código.

Para que esto sea posible hace falta un intermediario, un programa encargado de traducir cada instrucción escrita Código máquina (instrucciones de la CPU), este programa recibe el nombre de **intérprete** (en inglés *parser*).

El intérprete se encarga de leer una a una las instrucciones textuales del programa conforme estas necesitan ser ejecutadas y descomponerlas en instrucciones del sistema, además se encarga de automatizar algunas de las tareas típicas de un programador como declaraciones de variables o dependencias, de esta manera el proceso de programar se suele agilizar mucho lo cual repercute en la eficiencia del que tiene que escribir el código.

Ventajas

- **Independencia de la Plataforma:** Si queremos correr nuestro programa en otra computadora, no tenemos que preocuparnos de que la plataforma sea igual a la nuestra. Solo tenemos que asegurarnos que tenga instalado un intérprete para esa plataforma. Y si en un futuro se crea una nueva plataforma totalmente revolucionaria que tenga un intérprete de ese lenguaje nuestro programa no debería modificarse en una sola línea y seguir funcionando perfectamente.
- **Agilidad en el desarrollo:** Como no tenemos que compilar el programa para hacer pruebas (solo se debe correr en el parser), es más rápido para hacer pruebas y continuar con el desarrollo o corregir el error.

[3] Ver más https://www.python.org/dev/peps/pep-0008/

Desventajas

- **Dependencia del Intérprete:** El programa no puede correrse sin el intérprete.
- **Lentitud en la ejecución:** Como toda instrucción debe pasar por el intérprete, para que la traduzca y luego se ejecute, todo este proceso es, un poco, más lento que los programas compilados.

Multiplataforma

Significa que puede ser interpretado en diversos Sistemas Operativos como GNU/Linux, Windows, Mac OS, Solaris, entre otros.

Tipado Dinámico

Un lenguaje de tipado dinámico es aquel cuyas variables, no requieren ser definidas asignando su tipo de datos, sino que éste, se auto-asigna en tiempo de ejecución, según el valor declarado.

Multiparadigma

Acepta diferentes paradigmas (técnicas) de programación, tales como la orientación a objetos, aspectos, la programación imperativa y funcional.

Estructura de un Programa

La estructura de un programa en Python no es tan estricta como puede serlo en Pascal o en C/C++, ya que no debe comenzar con ninguna palabra reservada, ni con un procedimiento o función en particular. Simplemente con escribir un par de líneas de código ya podríamos decir que tenemos un programa en Python.

Lo que es importante destacar es la forma de identificar los distintos bloques de código. En Pascal se define un bloque de código usando las palabras reservadas Begin y End; en C/C++ se define mediante el uso de las llaves ({ y }). Sin embargo, en Python, se utiliza la indentación; es decir, la cantidad de espacios/tabs que hay entre el comienzo de la línea y el primer caracter distinto a ellos.

```
print('Ingrese 3 números distintos')
numUno = int(input())
numDos = int(input())
numTres = int(input())
if (numUno > numDos) & (numUno > numTres):
    if (numDos > numTres):
        print('De MAYOR a MENOR los nros son: ', str(numUno), '-', str(numDos),
'-', str(numTres))
    else:
        print('De MAYOR a MENOR los nros son: ', str(numUno), '-', str(numTres),
'-', str(numDos))

else:
    if (numDos > numUno) & (numDos > numTres):
        if numUno > numTres:
            print('De MAYOR a MENOR los nros son: ', str(numDos), '-',
str(numUno), '-', str(numTres))
    else:
        if (numTres > numUno) & (numTres > numDos):
            print('El mayor nro. ingresado es: ' + str(numTres))
```

REFERENCIAS

- Información disponible en http://web.fi.uba.ar/~bortega/algoritmos_I/descargas/apunte_python.pdf
- Información disponible en htttp://www.python.org
- "Python para principiantes", Eugenia Bahit. Creative Commons No comercial - Atribución - Compartir igual (CC BY-NC-SA) 3.0
- Información disponible en http://www.python.org.ar/wiki/AprendiendoPython
- Información disponible en http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf
- Casos de éxito: http://highscalability.com/youtube-architecture