El futuro digital es de todos

MinTIC

Universidad Industrial de Santander

MisiónTIC 2022

FUNDAMENTOS DEL LENGUAJE PYTHON

TEMA 4:
TIPOS DE DATOS E INSTRUCCIONES BÁSICAS

4.1. Intrudución

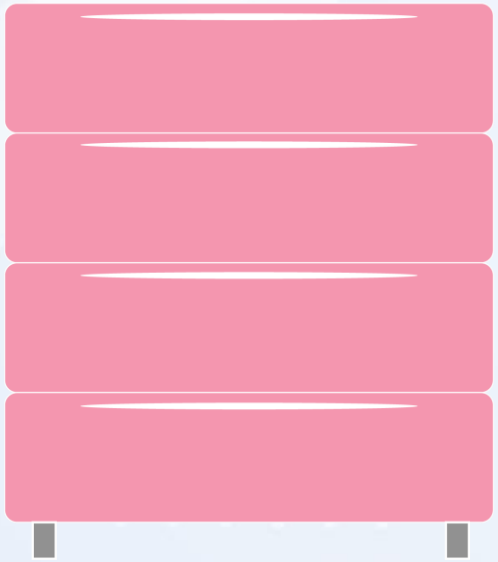
En Python hay varios tipos de datos compuestos estándar disponibles por defecto en el intérprete, como los tipos numéricos, secuencias, mapeos y conjuntos usados para agrupar otros valores.
Para el caso de las estructuras de datos se usan variables y constantes las cuales usan operadores para tratar los tipos de datos estándar. En este tema se describen las variables y los tipos de datos en el lenguaje Python.

4.2. Constantes y variables

En matemáticas llamamos constante a una magnitud que no cambia con el paso del tiempo.
En ocasiones, se puede tratar de un valor fijo y determinado.
Por otro lado, tenemos el concepto de variable, que se utiliza para definir toda cantidad susceptible de tomar distintos valores numéricos.
Python:

```
X = 5
Nombre = “Jairo”
Edad = 25 años.
```

Ejemplo:



ROPA	DÍA 1	DÍA 2	DÍA 3
Camisas	Azul	Verde	Blanco
Ropa Interior	Gris	Negra	Azul oscuro
Pantalones	Jeans	Drill Café	Tela azul
Accesorios	Correa negra	Correa negra	Correa negra

4.3. Declaración de variables

Para trabajar con variables se recomienda usar nombres descriptivos; si se quiere utilizar más de una palabra se debe escribir con guión bajo y finalmente cabe destacar que, la sintaxis correcta para utilizar el signo '=' es fijarse que siempre respetemos el utilizar un solo espacio después del signo igual.
NO se pueden usar:

- Espacios en blanco.
- Caracteres especiales como $ %&/()=?¿!,
- Comenzar una definición de variables con números.

4.4. Palabras reservadas

En los lenguajes informáticos, una palabra reservada es una palabra que tiene un significado gramatical especial para ese lenguaje y no puede ser utilizada como un identificador de objetos en códigos del mismo, como pueden ser las variables.

"and as assert break class continue def del elif else except exec finally for from global if import in is lambda not or pass print raise return try while with yield async, await, nonlocal, True, False y None"

4.5. Tipos fundamentales

En cualquier lenguaje de programación de alto nivel se manejan tipos de datos. Los tipos de datos definen un conjunto de valores que tienen una serie de características y propiedades determinadas.
Pensamos por un momento cuando éramos jóvenes y estábamos en el colegio en clase de matemáticas. Seguro que viste alguna clase en la que te enseñaban los distintos conjuntos de números. Los naturales (1, 2, 3, 4, …), los enteros (…, -2, -1, 0, 1, 2, …), los reales (… -1.1, -0.3, 2.1, …), etc; pues bien, en programación (y por supuesto en Python), cada uno de esos conjuntos sería lo que llamamos tipo de datos.
En Python, todo valor que pueda ser asignado a una variable tiene asociado un tipo de dato. Es importante entender que en Python todo es un objeto. Así que los tipos de datos serían las clases (donde se definen las propiedades y qué se puede hacer con ellas) y las variables serían las instancias (objetos) de los tipos de datos.

- Ejemplo de clase carro y la instancia Mercedes Benz SLC 500
- No te preocupes si no entiendes qué es una clase o un objeto, lo veremos más adelante.
- En definitiva, un tipo de dato establece qué valores puede tomar una variable y qué operaciones se pueden realizar sobre la misma.

Ver “Tipos de datos.pdf”

Los tipos de datos básicos de Python son los numéricos (enteros, punto flotante y complejos), las cadenas de caracteres y los booleanos.

Tipo Numéricos:

Enteros:

Son los números que no tienen decimales y pueden ser positivos y negativos (el 0 es un entero también).

Estos números son los conocidos int (entero) o long (entero largo para más precisión).

Ejem.:

X=-4

A = 54321

Reales:

Son los números que tienen decimales y son del tipo float.

X= 3.5502

Complejos:

Son los números que tienen una parte real y una imaginaria. Estos números se denominan complex y si no los conoces es probable que no lo necesites, aunque si te puedo decir que tienen algunas aplicaciones muy interesantes.

Ejemplo de número complejo:

X= 1,51 + 6j

Tipo Cadenas:

Las cadenas son texto encerrado entre comillas (simples o dobles) y se pueden conformar de diferentes tipos de caracteres (Numéricos, Alfabéticos, Especiales ‘#$%&*+’). Las cadenas admiten operadores como la suma o la multiplicación.

Ejemplo de cadenas:

```
cadena1 = 'comillas simples'
print (cadena1)

cadena2 = "comillas dobles"
print (cadena2)

n = "Aprender"
a = "Python"
na = n + " " + a
print (na)
```

Tipo Booleanos:

Este es el tipo de variable que solo puede tener un valor de Verdadero o Falso.
Son valores muy usados en condiciones y bucles.

```
aT = True

 print ("El valor es Verdadero:", aT, ", el cual es de tipo", type(aT)), "\n“

 aF = False

 print ("El valor es Falso:", aF, ", el cual es de tipo", type(aF))
```

Python también cuenta con tipos de datos complejos que admiten colecciones de datos: listas, tuplas y diccionarios.

Tipo Conjuntos:

Son una colección de datos sin elementos que se repiten:

```
pla = 'pastelito', 'jamon', 'papa', 'empanadilla', 'mango', 'quesito'
print (pla)
```

Tipo Listas:

Son listas que almacenan vectores (arrays). Estas listas pueden tener diferentes tipos de datos.

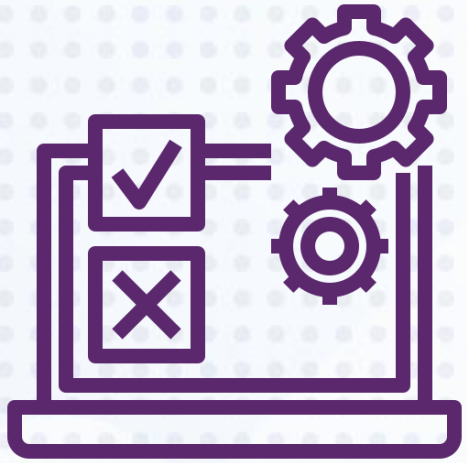
Ejemplo de listas en Python:

```
b = ['2.36', 'elefante', 1010, 'rojo']
print (b)
l4 = b[0:3:2]
print(l4)
```

Tipo Tuplas:

Es una lista que no se puede modificar después de su creación, es inmodificable:

```
# Ejemplo simple
    tupla = 1234, 5678, 'Hola Mundo Python!'
# Ejemplo tuplas anidadas
    otra = tupla, (1, 5, 3, 6, 5)
# operación asignación de valores de una tupla en variables
    x, y, z = tupla
    print(tupla)
    print(otra)
```



Tipo diccionarios:

Define los datos uno a uno entre un campo y un valor, ejemplo:

```
datos_basicos = {
        "nombres":"Pedro",
        "apellidos":"Perez Pinto",
        "numero":"852963",
        "fecha_nacimiento":"24/06/1990",
        "lugar_nacimiento":"San Vicente de Chucurí",
        "Departamento":"Santander",
        "estado_civil":"Casado"}
print ("\nDetalle del diccionario")
print ("==========================\n")
print ("\nClaves del diccionario:", datos_basicos.keys())
print ("\nValores del diccionario:", datos_basicos.values())
print ("\nElementos del diccionario:", datos_basicos.items())
#
print ("\nInscripcion de Curso")
print ("====================")
print ("\nDatos de participante")
print ("--------------------")
print ("Cedula de identidad:", datos_basicos['numero'])
print ("Nombre completo: " + datos_basicos['nombres'] + " " + datos_basicos['apellidos'])
```

Juguemos con >> type()

4.6. Asignación de expresiones

En Python las asignaciones no son expresiones, (una expresión es cualquier porción de código que retorne un valor) sino sentencias.
Para asignar un valor a una variable en Python se usa el signo "=", sin embargo, este tiene varias formas de uso.

Ver "Las expresiones de asignación.pdf"

Inicializar una variable antes de utilizarla:

```
edad = 18
print(edad)
```

Ahora se puede inicializar en una expresión que además es evaluada inmediatamente:

```
print(edad:=18)
```

Antes, para guardar el valor de una función se asignaba previamente a una variable y después esta se podía utilizar:

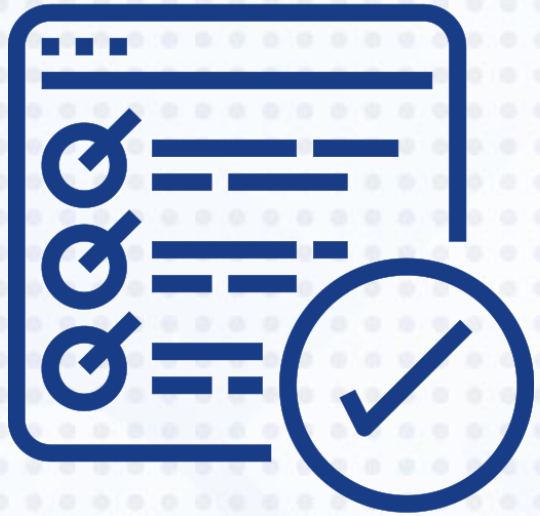
```
def media(nota1, nota2):
return (nota1 + nota2) / 2
notafinal = media(6, 8)
if notafinal >= 5:
    print('Aprobado con:', notafinal)
```

Ahora las dos acciones se pueden hacer en una misma línea:

```
if (notafinal:=media(6, 8)) >= 5:
    print('Aprobado con', notafinal)
```

Antes para hacer una asignación en un bucle se hacía dentro del mismo:

```
lista_compra = list()
while True:
    articulo = input('¿Qué necesitas comprar?: ')
    if articulo == '':
    print(lista_compra)
    break
    lista_compra.append(articulo)
```



Ahora la misma asignación se puede hacer en la misma línea de while:

```
lista_compra = list()
while (articulo := input('Qué necesitas comprar?: ')) != '':
    lista_compra.append(articulo)
print(lista_compra)
```

Las expresiones de asignación también se pueden utilizar en listas de comprensión:

```
parcelas_m2 = [220, 320, 180, 430]
precios = [(precio_m2:=100) * sup for sup in parcelas_m2]
print(f'{precios} al precio de {precio_m2} $ el m2')
# [22000, 32000, 18000, 43000] al precio de $100 el m2
```

El uso de paréntesis en las expresiones de asignación es fundamental para delimitar exactamente el valor que se asigna a una variable:
En el ejemplo que sigue a la variable no se asigna el valor 10, se asigna el resultado de comparar 10 y 5, es decir, True

```
if precio:=10 > 5:
    print(precio)  # True
```

En este caso queda más claro que es 10 el valor asignado a la variable precio:

```
if (precio:= 10) > 5:
print(precio)  # 10
```

Los paréntesis también permiten anidar expresiones para una asignación múltiple:

```
(total:= (precio:=10) * 5)
print(precio, total)  # 10
```

Pero atención, recuerda que existe una diferencia básica entre el uso de = y := en una asignación:
En el siguiente ejemplo se asigna una tupla de 3 valores a la variable:

```
var1 = 0, 1, 2
print(var1)
```

Y en la siguiente expresión se asigna sólo el primero de los valores: 0

```
(var1 := 0, 1, 2)
print(var1)
```

Lo recomendable en estos casos es delimitar también por paréntesis los valores de la tupla:

```
(var1 := (0, 1, 2))
print(var1)
```

Con delimitar solo los valores de la tupla no es suficiente. Hacerlo genera un error de sintaxis:

```
var1 := (0, 1, 2)
```

4.7. Transformación de datos

Python incluye una gran variedad de funciones nativas para realizar distintas operaciones. Por lo general, estas funciones vienen incorporadas dentro del lenguaje propiamente dicho; por eso, podemos tener la seguridad de que estarán disponibles en nuestra instalación. Para empezar, veamos cuáles de ellas son las que utilizaremos en este post:

1. Convertir a cadena de texto (string): str()

```
anios = 56
print = ("Edad: " + str(anios))
```

2. Convertir a entero: int()

```
Cant = "26"
costo = 1500 * int(Cant)
print (costo)
```

3. A punto flotante (números decimales): float()

   En caso que la función int reciba como argumento un número real, la parte decimal se perderá. Obsérvese también que float e int pueden recibir como argumento un texto conteniendo un número: Si se usa la función int el texto en cuestión deberá representar un entero pues, si se trata de un número real, devolverá un error.

La función float, por otro lado, puede extraer un número real de un texto tanto si éste representa un número real como si se trata de un número entero

```
Cant = "26.5"
costo = 1500 * float(Cant)
print (costo)
```

4. A booleano: bool()

   Esta función devuelve False si el elemento incluido como argumento es cero (si se trata de un número) o una cadena de texto vacía (si se trata de una cadena de texto). Si no es cero o no es una cadena de texto vacía, devuelve True

   ```
   bool(56) -> True
   bool("hola Mundo")  -> True
   bool(0)  -> False
   ```

Todas las funciones operan de la misma manera: esperan un argumento sobre el cual realizar la conversión.

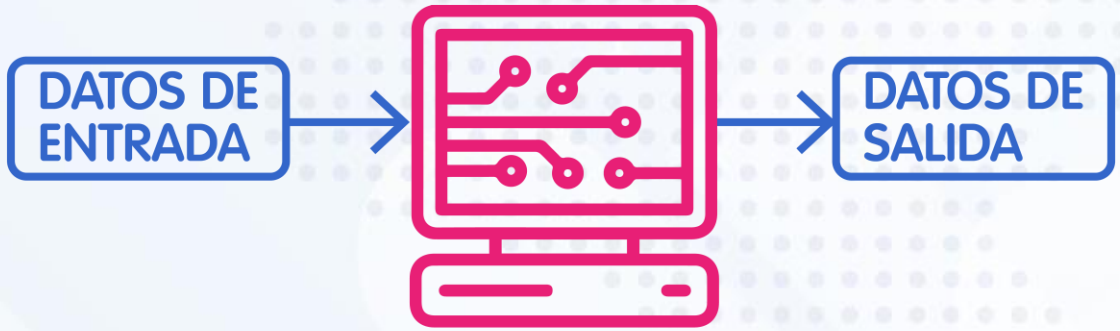
4.8. Entrada y salida estandar

Para pedir información al usuario, debe utilizar las funciones integradas en el intérprete del lenguaje, así como los argumentos de línea de comandos.

>> raw_input()

La forma general de mostrar información por pantalla es mediante una consola de comando, generalmente podemos mostrar texto y variables separándolos con comas, para este se usa la sentencia

>> print()



4.9. Bibliografía

- Introducción a la Programación con Python autor Andrés Marzal e Isabel Gracia
- sitio oficial: https://docs.python.org/3/