Números

Para empezar lo más sencillo es utilizar el intérprete de Python como una calculadora:

3 + 2

5

3 - 2

1

3 \* 2

6

Podemos utilizar comentarios # para explicar lo que hace nuestro código:

# División

3 / 2

1.5

# Módulo

3 % 2

1

# Potencia

3 \*\* 2

9

Podemos distinguir 2 tipos de números:

* **Enteros**: Que no tienen una parte decimal y van desde menos infinito a más infinito.
* **Flotantes** o decimales: Números que tienen una parte decimal escrita con un punto.

# Número entero

1

1

# Número flotante

323239829389.238273283

323239829389.2383

También podemos realizar operaciones más complejas. Python interpretará automáticamente las prioridades de los operadores:

3 - 2 + 4 \* 10

41

Variables

Este es el concepto más importante en la programación, así que asegúrate de entenderlo bien practicando mucho.

**Concepto fundamental**

Una variable es un identificador que representa un espacio en la memoria. A este espacio se le puede asignar un valor para utilizarlo posteriormente como si se tratase de un valor literal, incluso se puede operar con otras variables y reasignarle otro valor en cualquier momento.

Asignación de un valor a la variable 'n':

n = 3

n

3

Suma de una variable entera con un literal número entero:

n + 3

6

Producto de una variable entera con un literal número entero:

n \* 2

6

Producto de una variable entera con un literal número entero:

n \* n

9

Suma de dos variables enteras:

# Creamos otra variable

m = 10

n + m

13

Producto de dos variables y suma de un literal numérico:

n \* m + 10

40

Reasignación de los valores y cálculo de nuevo resultado:

n = 10

m = 15

n + m

25

Asignación del valor de una variable a otra:

n = m

n

15

Incluso se puede asignar una operación mezclada:

n = m + 10

n

25

O el propio valor sumado con un literal:

n = n + 25

n

50

Reutilización

Al crear una estructura de cálculos con variables podemos fácilmente adaptar sus valores para hacer distintas comprobaciones:

nota\_1 = 2

nota\_2 = 5

nota\_media = (nota\_1 + nota\_2) / 2

nota\_media

3.5

Cadenas de texto

Inmediatamente después de los números hay que echar un vistazo a las cadenas de texto, a fin de cuentas es la forma como las personas nos comunicamos de forma escrita. Las letras o caracteres son en definitiva símbolos de escritura y otro tipo de dato esencial.

Siempre se definen entre comillas simples o dobles:

'Hola Mundo'

'Hola Mundo'

"Hola Mundo"

'Hola Mundo'

Es posible poner comillas dobles en una cadena de comillas simples:

'Este texto incluye unas " "'

'Este texto incluye unas " "'

Comillas simples en una cadena de comillas dobles:

"Esta 'palabra' se encuentra escrita entre comillas simples"

"Esta 'palabra' se encuentra escrita entre comillas simples"

O también utilizar el carácter de escape \ para poner comillas del mismo tipo:

"Esta \"palabra\" se encuentra escrita entre comillas dobles"

'Esta "palabra" se encuentra escrita entre comillas dobles'

'Esta \'palabra\' se encuentra escrita entre comillas simples'

"Esta 'palabra' se encuentra escrita entre comillas simples"

# Métodos de las cadenas

## upper()

Devuelve la cadena con todos sus caracteres a mayúscula:

"Hola Mundo".upper()

'HOLA MUNDO'

## lower()

Devuelve la cadena con todos sus caracteres a minúscula:

"Hola Mundo".lower()

'hola mundo'

## capitalize()

Devuelve la cadena con su primer carácter en mayúscula:

"hola mundo".capitalize()

'Hola mundo'

## title()

Devuelve la cadena con el primer carácter de cada palabra en mayúscula:

"hola mundo".title()

'Hola Mundo'

## count()

Devuelve una cuenta de las veces que aparece una subcadena en la cadena:

"Hola mundo".count('mundo')

1

## find()

Devuelve el índice en el que aparece la subcadena (-1 si no aparece):

"Hola mundo".find('mundo')

5

"Hola mundo".find('mundoz')

-1

## rfind()

Devuelve el índice en el que aparece la subcadena, empezando por el final:

"Hola mundo mundo mundo".rfind('mundo')

17

## isdigit()

Devuelve True si la cadena es todo números (False en caso contrario):

c = "100"

c.isdigit()

True

## isalnum()

Devuelve True si la cadena es todo números o carácteres alfabéticos:

c = "ABC10034po"

c.isalnum()

True

## isalpha()

Devuelve True si la cadena es todo carácteres alfabéticos:

c = "ABC10034po"

c.isalpha()

False

"Holamundo".isalpha()

True

## islower()

Devuelve True si la cadena es todo minúsculas:

"Hola mundo".islower()

False

## isupper()

Devuelve True si la cadena es todo mayúsculas:

"Hola mundo".isupper()

False

## istitle()

Devuelve True si la primera letra de cada palabra es mayúscula:

"Hola Mundo".istitle()

True

## isspace()

Devuelve True si la cadena es todo espacios:

" - ".isspace()

False

## startswith()

Devuelve True si la cadena empieza con una subcadena:

"Hola mundo".startswith("Mola")

False

## endswith()

Devuelve True si la cadena acaba con una subcadena:

"Hola mundo".endswith('mundo')

True

## split()

Separa la cadena en subcadenas a partir de sus espacios y devuelve una lista:

"Hola mundo mundo".split()[0]

'Hola'

Podemos indicar el carácter a partir del que se separa:

"Hola,mundo,mundo,otra,palabra".split(',')

['Hola', 'mundo', 'mundo', 'otra', 'palabra']

## join()

Une todos los caracteres de una cadena utilizando un caracter de unión:

",".join("Hola mundo")

'H,o,l,a, ,m,u,n,d,o'

" ".join("Hola")

'H o l a'

## strip()

Borra todos los espacios por delante y detrás de una cadena y la devuelve:

" Hola mundo ".strip()

'Hola mundo'

Podemos indicar el carácter a borrar:

"-----Hola mundo---".strip('-')

'Hola mundo'

## replace()

Reemplaza una subcadena de una cadena por otra y la devuelve:

"Hola mundo".replace('o','0')

'H0la mund0'

Podemos indicar un límite de veces a reemplazar:

"Hola mundo mundo mundo mundo mundo".replace(' mundo','',4)

'Hola mundo'

La función print()

Es una instrucción que nos permite mostrar correctamente el valor de una cadena (u otros valores/variables) por pantalla:

"Una cadena"

'otra cadena'

'otra cadena más'

'otra cadena más'

print("Una cadena")

print('otra cadena')

print('otra cadena más')

Una cadena

otra cadena

otra cadena más

Acepta caracteres especiales como las tabulaciones \t o los saltos de línea \n:

print("Un texto\tuna tabulación")

Un texto una tabulación

print("Un texto\nuna nueva línea")

Un texto

una nueva línea

Para evitar los caracteres especiales, debemos indicar que una cadena es cruda (raw):

print("C:\nombre\directorio")

C:

ombre\directorio

print(r"C:\nombre\directorio") # r(cadena) => raw (cruda)

C:\nombre\directorio

Podemos utilizar """ *(triple comillas)* para cadenas multilínea:

print("""Una línea

otra línea

otra línea\tuna tabulación""")

Una línea

otra línea

otra línea una tabulación

También es posible asignar cadenas a variables. La forma correcta de mostrarlas es con la instrucción *print()*:

c = "Esto es una cadena\ncon dos líneas"

c

'Esto es una cadena\ncon dos líneas'

c = "Esto es una cadena\ncon dos líneas"

print(c)

Esto es una cadena

con dos líneas

Operaciones

Una de las operaciones de las cadenas es la concatenación (o suma de cadenas):

c = "Esto es una cadena\ncon dos líneas"

c + c

'Esto es una cadena\ncon dos líneasEsto es una cadena\ncon dos líneas'

c = "Esto es una cadena\ncon dos líneas"

print(c+c)

Esto es una cadena

con dos líneasEsto es una cadena

con dos líneas

s = "Una cadena" " compuesta de dos cadenas"

print(s)

Una cadena compuesta de dos cadenas

c1 = "Una cadena"

c2 = "otra cadena"

print("Una cadena " + c2)

Una cadena otra cadena

También es posible utilizar la multiplicación de cadenas:

diez\_espacios = " " \* 10

print(diez\_espacios + "un texto a diez espacios")

un texto a diez espacios

Índices en las cadenas

Los índices nos permiten posicionarnos en un carácter específico de una cadena.

Representan un número [índice], que empezando por el 0 indica el carácter de la primera posición, y así sucesivamente:

palabra = "Python"

palabra[0] # carácter en la posición 0

'P'

palabra[3]

'h'

El índice negativo -1, hace referencia al carácter de la última posición, el -2 al penúltimo y así sucesivamente:

palabra[-1]

'n'

print(palabra[-0])

print(palabra[-2])

print(palabra[-6])

P

o

P

Slicing en las cadenas

El slicing es una capacidad de las cadenas que devuelve un subconjunto o subcadena utilizando dos índices [inicio:fin]:

* El primer índice indica donde empieza la subcadena (se incluye el carácter).
* El segundo índice indica donde acaba la subcadena (se excluye el carácter).

palabra = "Python"

palabra[0:2]

'Py'

palabra[2:]

'thon'

palabra[-2:]

'on'

Si en el slicing no se indica un índice se toma por defecto el principio y el final:

palabra[:2]

'Py'

palabra[2:]

'thon'

palabra[:]

'Python'

palabra[:2] + palabra[2:]

'Python'

Si un índice se encuentra fuera del rango de la cadena, dará error:

palabra[99]

---------------------------------------------------------------------------

IndexError Traceback (most recent call last)

<ipython-input-47-b31ddef6ab27> in <module>()

----> 1 palabra[99]

IndexError: string index out of range

Pero con slicing esto no pasa y simplemente se ignora el espacio hueco:

palabra[:99]

'Python'

palabra[99:]

''

Inmutabilidad

Una propiedad de las cadenas es que no se puede modificar su contenido. Si intentamos reasignar un carácter, no nos dejará:

palabra[0] = "N"

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

<ipython-input-51-c87a9e773639> in <module>()

----> 1 palabra[0] = "N"

TypeError: 'str' object does not support item assignment

Sin embargo, utilizando slicing y concatenación podemos generar nuevas cadenas fácilmente:

palabra = "N" + palabra[1:]

palabra

'Nython'

Funciones

Un ejemplo de función útil que soportan las cadenas es *len()*, que nos permite saber su longitud (el número de caracteres que contienen):

palabra = "Python"

len(palabra)

6

Hay más funciones, pero las iremos descubriendo a lo largo del curso.

-----------------------------------------------------------------------------------------------------------------

# Listas

Las listas se tratan de un tipo compuesto de dato que puede almacenar distintos valores (llamados ítems o elementos) ordenados entre [ ] y separados con comas:

numeros = [1,2,3,4]

numeros

[1, 2, 3, 4]

## Índices y slicing

Funcionan de una forma muy similar a las cadenas de caracteres:

datos = [4, "Una cadena", -15, 3.14, "Otra cadena"]

print(datos[0])

print(datos[-1])

print(datos[2:])

4

Otra cadena

[-15, 3.14, 'Otra cadena']

## Suma de listas

Da como resultado una nueva lista que incluye todos los ítems:

numeros + [5,6,7,8]

[1, 2, 3, 4, 5, 6, 7, 8]

## Mutabilidad

A diferencia de las cadenas, en las listas sí podemos modificar sus ítems utilizando índices:

pares = [0,2,4,5,8,10]

pares[3] = 6

pares

[0, 2, 4, 6, 8, 10]

Integran funcionalidades internas como el método .append() para añadir un ítem al final de la lista:

pares.append(12)

pares.append(7\*2)

pares

[0, 2, 4, 6, 8, 10, 12, 14]

Y una peculiaridad es que también aceptan asignación con slicing para modificar varios ítems en conjunto:

letras = ['a','b','c','d','e','f']

letras[:3]

['a', 'b', 'c']

letras[:3] = ['A','B','C']

letras

['A', 'B', 'C', 'd', 'e', 'f']

Asignar una lista vacía equivale a borrar los ítems de la lista o sublista:

letras[:3] = []

letras

['d', 'e', 'f']

letras = []

letras

[]

La función len() funciona con las listas del mismo modo que en las cadenas:

print(len(letras))

print(len(pares))

0

8

## Listas anidadas

Podemos manipular fácilmente este tipo de estructuras utilizando múltiples índices, como si nos refieréramos a las filas y columnas de una tabla:

a = [1,2,3]

b = [4,5,6]

c = [7,8,9]

r = [a,b,c]

r

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(r[0]) # Primera sublista

print(r[-1]) # Última sublista

print(r[0][0]) # Primera sublista, y de ella, primer ítem

print(r[1][1]) # Segunda sublista, y de ella, segundo ítem

print(r[2][2]) # Tercera sublista, y de ella, tercer ítem

print(r[-1][-1]) # Última sublista, y de ella, último ítem

[1, 2, 3]

[7, 8, 9]

1

5

9

9

# Métodos de las listas

## append()

Añade un ítem al final de la lista:

lista = [1,2,3,4,5]

lista.append(6)

lista

[1, 2, 3, 4, 5, 6]

## clear()

Vacía todos los ítems de una lista:

lista.clear()

lista

[]

## extend()

Une una lista a otra:

l1 = [1,2,3]

l2 = [4,5,6]

l1.extend(l2)

l1

[1, 2, 3, 4, 5, 6]

## count()

Cuenta el número de veces que aparece un ítem:

["Hola", "mundo", "mundo"].count("Hola")

1

## index()

Devuelve el índice en el que aparece un ítem (error si no aparece):

["Hola", "mundo", "mundo"].index("mundo")

1

## insert()

Agrega un ítem a la lista en un índice específico:

Primera posición (0):

l = [1,2,3]

l.insert(0,0)

l

[0, 1, 2, 3]

Penúltima posición (-1):

l = [5,10,15,25]

l.insert(-1,20)

l

[5, 10, 15, 20, 25]

Última posición en una lista con len():

l = [5,10,15,25]

n = len(l)

l.insert(n,30)

l

[5, 10, 15, 20, 25, 30]

Una posición fuera de rango añade el elemento al final de la lista (999):

l.insert(999, 35)

l

[5, 10, 15, 20, 25, 30, 35]

## pop()

Extrae un ítem de la lista y lo borra:

l = [10,20,30,40,50]

print(l.pop())

print(l)

50

[10, 20, 30, 40]

Podemos indicarle un índice con el elemento a sacar (0 es el primer ítem):

print(l.pop(0))

print(l)

10

[20, 30, 40]

## remove()

Borra el primer ítem de la lista cuyo valor concuerde con el que indicamos:

l = [20,30,30,30,40]

l.remove(30)

print(l)

[20, 30, 30, 40]

## reverse()

Le da la vuelta a la lista actual:

l.reverse()

print(l)

[40, 30, 30, 20]

Las cadenas no tienen el método .reverse() pero podemos simularlo haciendo unas conversiones:

lista = list("Hola mundo")

lista.reverse()

cadena = "".join(lista)

cadena

'odnum aloH'

## sort()

Ordena automáticamente los ítems de una lista por su valor de menor a mayor:

lista = [5,-10,35,0,-65,100]

lista.sort()

lista

[-65, -10, 0, 5, 35, 100]

Podemos utilizar el argumento reverse=True para indicar que la ordene del revés:

lista.sort(reverse=True)

lista

[100, 35, 5, 0, -10, -65]

Ejercicio 1

Identifica el tipo de dato (int, float, string o list) de los siguientes valores literales:

"Hola Mundo"

[1, 10, 100]

-25

1.167

["Hola", "Mundo"]

' '

Ejercicio 2

Determina mentalmente (sin programar) el resultado que aparecerá por pantalla en las siguientes operaciones con variables:

a = 10

b = -5

c = "Hola "

d = [1, 2, 3]

print(a \* 5)

print(a - b)

print(c + "Mundo")

print(c \* 2)

print(d[-1])

print(d[1:])

print(d + d)

Ejercicio 3

El siguiente código pretende realizar una media entre 3 números, pero no funciona correctamente. ¿Eres capaz de identificar el problema y solucionarlo?

numero\_1 = 9

numero\_2 = 3

numero\_3 = 6

media = numero\_1 + numero\_2 + numero\_3 / 3

print("La nota media es", media)

Ejercicio 4

A partir del ejercicio anterior, vamos a suponer que cada número es una nota, y lo que queremos es obtener la nota final. El problema es que cada nota tiene un valor porcentual:

* La primera nota vale un 15% del total
* La segunda nota vale un 35% del total
* La tercera nota vale un 50% del total

Desarrolla un programa para calcular perfectamente la nota final:

nota\_1 = 10

nota\_2 = 7

nota\_3 = 4

# Completa el ejercicio aquí

Ejercicio 5

La siguiente matriz (o lista con listas anidadas) debe cumplir una condición, y es que en cada fila el cuarto elemento siempre debe ser el resultado de sumar los tres primeros. ¿Eres capaz de modificar las sumas incorrectas utilizando la técnica del slicing?

**Ayuda**

La función llamada *sum(lista)* devuelve una suma de todos los elementos de la lista ¡Pruébalo!

matriz = [

[1, 1, 1, 3],

[2, 2, 2, 7],

[3, 3, 3, 9],

[4, 4, 4, 13]

]

# Completa el ejercicio aquí

------------------------------------------------------------------------------------------------------------------------------