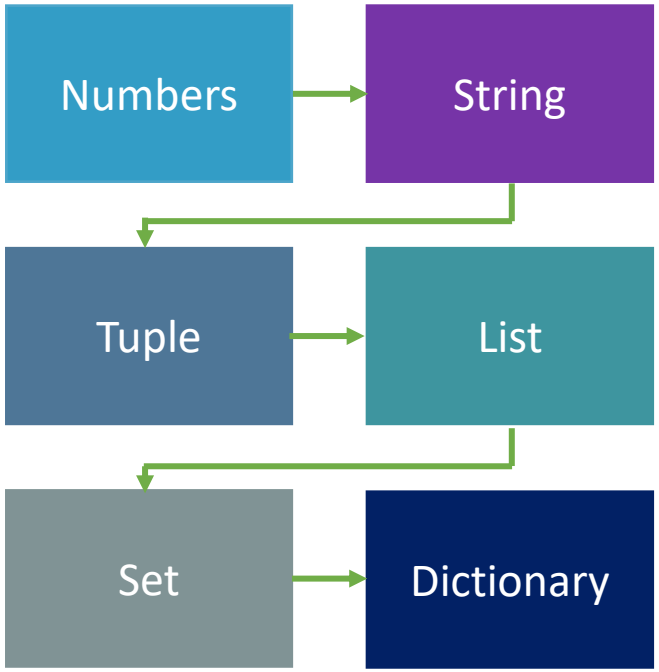
Jul. 2022

Unidad 2: Tipos de datos



Introducción

- Clasifican e Indican al intérprete cómo el programa pretende utilizar esos datos
  - las diferentes operaciones que se pueden realizar

Objetivos de aprendizaje

Ser capaz de

- Explicar los diferentes tipos de datos
- Utilizar operadores según el tipo de dato
- Implementar operaciones
- Describir las secuencias de escape

Datos numéricos

- Tipos de números
  - Enteros
  - Decimales (coma flotante)

Números enteros

- Los números enteros pueden ser negativos o positivos o cero, y nunca una fracción

```
>>> entero = 49
>>> entero_negativo = -35
```

- Precisión ilimitada
  - no hay límites en cuanto a su tamaño
  - salvo la memoria disponible

```
>>> entero_largo = 12345678983274638932168475321490225636477542278854390166621455533643278899854210000
```

Números en coma flotante

- El tipo para este tipo de valores es float.

```
>>> n = 3.3333
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.7182818459045
```

- Convertir un número entero en un valor de coma flotante

```
>>> float(23)
```

Números binarios, hexadecimales y octales

- Son sistemas numéricos alternativos
- Expresados en el sistema de base (2, 16, 8)
- Para representar números utiliza:
  - [B] sólo 0s y 1s
  - [H] símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e y f
  - [O] dígitos del 0 al 7
- En Python escribe el número y se pone el prefijo
  - 0b
  - 0x
  - 0o
- Al escribir un número de este tipo en el intérprete se obtiene su equivalente decimal
- Para convertir cualquier número decimal (base 10) a estos números
  - Funciones bin, hex y oct

```
>>> 0b
7
>>> 0b10
2
```

```
>>> 0xf
15
>>> 0x9ac
2476
```

```
>>> 0o20
16
>>> 0o200
128
```

E1: Conversión entre diferentes tipos de sistemas numéricos

- Crear un script que tome la entrada del usuario y la convierta en un número binario

- 1. Definir la variable número que toma la entrada del usuario
- 2. Convertir la entrada a un número entero
- 3. Convertir el numero entero en un número binario
- 4. Imprime el valor

Salida de ejemplo:

```
Numero para convertir a binario: 5
0b101
```

Operadores

- Aritméticos
- De asignación

Operadores aritméticos

- Son funciones matemáticas que toman y realizan cálculos sobre los valores numéricos
- Todos los tipos numéricos de Python soportan las siguientes operaciones:
  - Suma:
  - Resta:
  - Multiplicación:
  - División:
  - División de enteros:
  - Módulo:
  - Exponenciación:

```
>>> 5 +  8 + 7
>>> 20 - 5
>>> 4 * 3
>>> 12 / 3
>>> 13 // 2
>>> 5 % 2
>>> 5 ** 3
```

(da lugar a un número de punto flotante)
(= 6)
(el resto)
(Eleva un número a una potencia)

Operadores aritméticos

Operator	Result
x + y	Sum of x and y
x - y	Difference of x and y
x * y	Product of x and y
x / y	Quotient of x and y
x // y	Floored quotient of x and y
x % y	Remainder of x and y
-x	x negated
+x	x unchanged
abs(x)	Absolute value or magnitude of x
int(x)	x converted to integer
float(x)	x converted to floating point
divmod(x, y)	Returns the pair (x // y, x % y)
pow(x, y)	x to the power y
x ** y	x to the power y

Operadores de asignación

- El operador de asignación simple =
- Python tiene otros operadores de asignación
  - variaciones abreviadas de los operadores simples
  - no sólo realizan una operación aritmética
    sino que también reasignan la variable

```
>>> x = 10
>>> x += 1
>>> print(x)
11
```

Lista de todos los operadores de asignación

- El operador realiza la operación a la variable del lado izquierdo el valor del lado derecho

Operator	Example	Equivalent to
+=	x += 7	x = x + 7
-=	x -= 7	x = x - 7
*=	x *= 7	x = x * 7
/=	x /= 7	x = x / 7
%=	x %= 7	x = x % 7
**=	x **= 7	x = x ** 7
//=	x //= 7	x = x // 7

Orden de operaciones

- El conjunto de reglas sobre qué procedimientos deben ser evaluados primero al evaluar una expresión
- El orden en el que se evalúan los operadores es PEMDAS

  matemáticamente se evalúan:

  1. Las expresiones dentro de los paréntesis primero
  2. Exponenciación
  3. La multiplicación y la división (incluyendo división de piso y el módulo)
  4. La suma y la resta
- Las sentencias se evalúan de izquierda a derecha

Actividad 1: Orden de las operaciones

- Reescribe la siguiente ecuación como una expresión de Python y obtén el resultado

$$5(4-2)+\left(\frac{100}{\frac{5}{2}}\right)2$$

Actividad 2: Uso de operadores aritméticos

- Escribir un script que tome como entrada del usuario el numero de días y lo convierta en años, semanas y días (ignorar los años bisiestos)

```
Número de dias: 600
años: 1
semanas: 33
dias: 4
```

1. Declarar la entrada del usuario
2. Convertir a un número entero
3. Calcular el número de años en ese conjunto de días
4. Los días restantes que no fueron convertidos a años convertirlo en semanas
5. Los días restantes que no fueron convertidos a semanas dejarlo en días
6. Imprimir el resultado

Cadenas de caracteres *(strings)*

- Las cadenas son una secuencia de caracteres
- Pueden ir entre comillas simples (‘) o dobles (")

```
>>> "una cadena"
'una cadena'
>>> ‘otra cadena’
‘otra cadena’
```

- Una cadena con comillas dobles puede contener comillas simples y viceversa

```
>>> “Car’s”
“Car's"
```

```
>>> '"¡Socorro!", exclamó’
'"¡Socorro!", exclamó.’
```

Cadenas de caracteres *(strings)*

- Entre comillas triples se encierra los caracteres de una cadena multilínea

```
>>> s = """Esta es una cadena multilínea
... 1
... 2
... 3
... fin de la cadena multilínea"""
>>> s
'Esta es una cadena multilínea\n1\n2\n3\nfin
de la cadena multilínea'
```

Operaciones con cadenas de caracteres

1. Repetir cadenas (operador *)

```
>>> print(“-” * 25)
```

2. Concatenar cadenas (operador +)

```
>>> "I "+"love "+"Python"
'I love Python’
```

- Las cadenas son inmutables

```
>>> hola = Hola
>>> mundo = Mundo
>>> hola = hola + mundo
Hola Mundo'
```

Operaciones con cadenas de caracteres

3. Indexación

- de izquierda a derecha
- de derecha a izquierda

0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
I		l	o	v	e		P	y	t	h	o	n

```
>>> s="I love Python"
>>> s[0]
'I'
>>> s[10]
'h'
>>> s[-10]
'o'
```

* Un índice con un carácter que no existe Python lanzá un IndexError.

Operaciones con cadenas de caracteres

4. Obtener una **rebanada/subcadena** dentro de un rango de índices

```
>>> cadena = "Bioelectromagnetismo"
>>> cadena[0:3]
'Bio’
>>> cadena[:3]
'Bio'
>>> cadena[10:20]
'magnetismo’
>>> cadena[10:]
'magnetismo'
>>> cadena[3:10]
'electro’
>>> cadena[-17:-10]
'electro'
```

cadena[índice_inicial : índice_final]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
B	i	o	e	l	e	c	t	r	o	m	a	g	n	e	t	i	s	m	o

Actividad 3: Rebanar cadenas (subcadenas)

- Dadas las siguientes sentencias ¿cuál será la salida?

```
1. >>> "[8,9,10]"[3]
2. >>> "La curiosidad por explorar la vida"[7]
3. >>> "La curiosidad por explorar la vida"[12:19]
4. >>> "más allá de las estructuras"[9:]
5. >>> "más allá de las estructuras"[:-1]
```

Operaciones con cadenas de caracteres

5. Longitud = número de caracteres que contiene

- utilizando la función len()
- toma una cadena como parámetro y devuelve un número entero

```
>>> pregunta = "¿Quién fue el primero de los Beatles que dejó el grupo?"
>>> len(pregunta)
55
```

6. Formato: construir nuevas cadenas utilizando valores existentes

6.1 Interpolación de cadenas

6.2 Método str.format()

6.3 Formato %.

Operaciones con cadenas de caracteres

6.1 Interpolación de cadenas

- Utiliza cadenas formateadas
  - Van precedidas de una **f** para indicar cómo deben interpretarse
- Para insertar una variable, se colocan llaves que contienen la expresión que queremos poner dentro de la cadena

```
>>> pastel=‘helado’
>>> f"Comí un poco de {pastel} y estaba buenísimo"
‘Comí un poco de helado y estaba buenísimo’
```

```
>>> numero=7
>>> f"{numero*2} es sólo un número"
'14 es sólo un numero'
```

Operaciones con cadenas de caracteres

6.2 El método str.format()

- Ponemos llaves en las posiciones donde queremos poner nuestros valores
- Llamamos al método format
  - toma el argumento (nuestra variable)
  - sustituye las llaves por el valor

```
>>> reduccion=43
>>> año=2030
>>> cadena="En el año {} las emisiones se reducirán un {}%".format(año,reduccion)
>>> cadena
'En el año 2030 las emisiones se reducirán un 43%'
```

Operaciones con cadenas de caracteres

6.3 Formateo de operador %

- Al estilo de formateo % del lenguaje C.
- Se utiliza el carácter % seguido del especificador de formato

  %s para caracteres

  %d para enteros
- Este método es inflexible y es más difícil de usar correctamente

```
>>> numero=3
>>> mascotas="gatos"
>>> "Tengo %d %s" %(numero, mascotas)
'Tengo 3 gatos'
```

Métodos con cadenas de caracteres

Método:	**devuelve una copia de la cadena con:**
• str.capitalize()	la primera letra en mayúscula y el resto en minúscula
• str.lower()	todos los caracteres convertidos a minúsculas
• str.upper()	todos los caracteres convertidos a mayúsculas:
• str.startswith()	comprueba si una cadena empieza por el prefijo especificado
• str.endswith()	comprueba que la cadena termina con el sufijo especificado
• str.strip()	Elimina los caracteres iniciales y finales sin argumento -> elimina todos los espacios en blanco
• str.replace()	sustituye todas las apariciones por la nueva

```
>>> "---mi editor favorito es vi----------".strip("-")
'mi editor favorito es vi’
>>> "mi lenguaje favorito es Java".replace(“Java","Python")
'mi lenguaje favorito es Python'
```

Actividad 4: Trabajar con cadenas

- Escriba un script que convierta las últimas n letras de una cadena a mayúsculas
  debe tomar como entrada del usuario
  a. la cadena a convertir y
  b. un número entero, que especifica las últimas n letras a convertir

1. Solicitar al usuario la cadena a convertir
2. Solicitar el número de últimas letras a convertir
3. Dividir la cadena, y obtener la primera parte
4. Obtener la última parte de la cadena (la que vamos a convertir)
5. Transformar la subcadena
6. Concatenar la primera y la nueva última parte
7. Ejecuta el script

- La salida debería ser algo parecido a esto:

```
Cadena a convertir: tipos de datos compuestos
¿Cuántas últimas letras hay que convertir? 5
tipos de datos compuESTOS
```

Secuencias de escape

- Es una secuencia de caracteres que no representa su significado literal cuando está dentro de una cadena
- El carácter de escape es la barra invertida \.

```
>>> print("Hola\nMundo")
Hola
Mundo
```

```
>>> print('Pepe\'s Bar')
"Pepe's Bar"
```

Lista completa de secuencias de escape

Escape Sequence	Definition
\newline	Backslash and newline ignored
\\	Backslash (\)
\’	Single quote (‘)
\”	Double quote (“)
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)
\ooo	Character with octal value ooo
\xhh	Character with hex value hh

Ejercicio 2: Uso de secuencias de escape

Escribiremos un script de Python que descompone una frase por palabras e imprime cada palabra en su propia línea

1. Solicita al usuario la frases a dividir
2. Sustituye todos los espacios por caracteres de nueva línea
3. Imprime la frase

La salida debería ser algo parecido a esto:

```
Cadena a dividir: hola que tal
hola
que
tal
```

Actividad 5: Manipulación de Cadenas

- Escriba un script que cuente el número de ocurrencias de una palabra especificada en una frase dada

1. Tomar de la entrada del usuario la frase y la palabra a buscar
2. Formatear la entrada eliminando los espacios en blanco y convirtiéndola en minúsculas
3. Cuente las ocurrencias de la subcadena
4. Imprime los resultados

La salida debería ser algo parecido a esto:

```
Frase: En medio del camino de nuestra vida me encontré en un oscuro bosque
Palabra a buscar en la frase: en
Hay 3 ocurrencias de 'en' en la frase.
```

Listas

- Los arrays se conocen como listas
- Las listas son un tipo de datos agregados
  - se componen de otros tipos de datos
- Las listas son similares a las cadenas
  - los valores dentro de ellas están indexados
  - tienen una propiedad de longitud y un conteo
- Las listas son heterogéneas
  - pueden contener valores de diferentes tipos
- Las listas son mutables
  - puedes cambiar los valores dentro de ellas sobre la marcha

Listas

- Las listas se hacen con elementos separados por comas y encerrados entre corchetes

```
>>> digitos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> letras = ["a", "b", "c", "d"]
>>> lista_mixta = [1, 3.14159, "Primavera", "Verano", [1, 2, 3, 4]]
```

- Las listas pueden indexarse (igual que las cadenas)
  - El primer elemento de una lista comienza en el índice 0:

Operaciones con listas

- Tamaño — len() obtiene el número de elementos de una lista
- Trocear — devuelve una nueva lista derivada de la anterior
- Concatenar — puedes sumar dos listas utilizando el operador +
- Cambiar valores — asignar el nuevo valor a lo que esté en ese índice
- Insertar — insertar un valor al final de una lista con list.append()

Ejercicio 3: Referencias a listas

- Cree una nueva lista

```
>>> lista_1 = [1, 2, 3]
```

- Luego, asigna una nueva variable, lista_2 a

```
>>> lista_2 = lista_1
```

- Añade 4 a la lista_2 y comprueba el contenido de la lista_1

```
>>> lista_2.append(4)
>>> lista_1
[1, 2, 3, 4]
```

- Inserta el valor ‘a’ en el índice 0 de la lista_1

```
>>> lista_1[0] = "a"
```

- Comprueba el contenido de la lista_2

```
>>> lista_2
['a', 2, 3, 4]
```

Actividad 6: Trabajar con listas

Escribir un script que obtenga los n primeros elementos de una lista

1. Crea la lista con los siguientes elementos: 57, 16, 37, 83, 571, 163, 9, 41, 27
2. Imprimir la lista
3. Leer la entrada del usuario con el número de elementos a obtener
5. Imprimir la porción de la lista desde el primer hasta el n-elemento
6. Ejecute el script

- La salida debería ser la siguiente

```
Lista:  [57, 16, 37, 83, 571, 163, 9, 41, 27]
Número de elementos a recuperar de la lista: 3
[57, 16, 37]
```

Ejercicio 3: Ejercicios con listas

1. Lista vacía
2. Insertamos elementos
3. Llamar elemento
4. Lista de listas
5. Rebanar
6. Obtener la longitud
7. cambiar los valores de los elementos
8. eliminar
9. Pertenencia
10. Asignación de un elemento de la lista a una variable
11. Encontrar valores index()
12. Añadir valores a una lista con append(), e insert()
13. Eliminar un elemento
14. Ordenar elementos

Booleanos

- Son valores que sólo pueden ser uno de dos valores
  - Verdadero o Falso

```
>>> True
True
>>> type(True)
<class 'bool'>
```

- Los booleanos se asocian con las sentencias de control
  - cambian el flujo del programa
  - dependiendo de la veracidad de las cantidades

Operadores de comparación

- Comparan los valores de los objetos, los objetos, las identidades
- Los objetos no necesitan ser del mismo tipo.
- Hay ocho operadores de comparación en Python:

Operator	Meaning
<	Less than or equal to
>	Greater than
>=	Greater than or equal
==	Equal to
!=	Not equal to
is	Object identity
is not	Negated object identity

Operadores Lógicos

- Combinan expresiones booleanas

Operator	Result
not x	Returns false if x is true, else false
x and y	Returns x if x is false, else returns y
x or y	Returns y if x is false, else returns x

- **and** sólo evalúa el segundo argumento SI el primero es Verdadero
- **or** sólo evalúa el segundo argumento SI el primero es Falso

Operadores de pertenencia

- Los operadores **in** y **not in** comprueban la pertenencia
- Todas las secuencias (listas y cadenas) admiten este operador
- Recorren cada elemento para ver si el elemento que se busca está dentro de la lista
- Los valores de retorno son True o False

```
>>> numeros = [1,2,3,4,5]
>>> 3 in numeros
True
>>> 6 in numeros
False
```

Tuplas

- Son listas inmutables
- Se denotan utilizando paréntesis en lugar de los corchetes
- Creación

```
>>> deportes1 = ('fútbol' , 'natación' , 'ciclismo')
>>> deportes2 = 'fútbol' , 'natación' , 'ciclismo’
```

- Operaciones

```
>>> deportes1 + deportes2
>>> deportes3=(deportes1, deportes2)
>>> deportes1=deportes1*3
>>> deportes1[2]
>>> deportes1[2]='running'
>>> deportes1=deportes1[2:4]
>>> del deportes1
```

Diccionarios

- Permiten acceder a cualquier valor utilizando la clave
- Se indexan mediante claves (cadenas)
- Hay dos tipos de diccionarios
  - Dict: que no está ordenado,
  - OrderedDict: mantiene el orden de inserción.

Creación de un diccionario

- Dos maneras:
  - Usando llaves
  - Utilizar la función dict()

```
>>> diccionario = {}
>>> diccionario = dict()
```

```
>>> d1 = {"comunidad":"Cataluña", "ciudad":"Barcelona"}
>>> d2 = dict(comunidad=“Galicia", ciudad=“Santiago")
```

Añadir datos a un diccionario

```
>>> d1["población"]=“1 millon"
```

```
>>> d1["población"]=“1.5 millones"
```

Lectura de datos de un diccionario

```
>>> d1["ciudad"]
```

```
>>> d1["comunidad"]
```

```
>>> d1["población"]
```

```
>>> print(d1.get("población"))
```

Iterando a través de los diccionarios

```
>>> for elemento in d1:
        print(elemento)
```

```
>>> for elemento in d1.keys():
        print(elemento)
```

```
>>> for elemento in d1.values():
        print(elemento)
```

```
>>> for elemento in d1.items():
        print(elemento)
```

Comprobación de claves concretas

```
>>> "ciudad“ in d1
```

```
>>> “poblacion“ in d1
```

Atributos adicionales del diccionario

```
>>> b={}
```

```
>>> b.update({"nombre":"Juan Perez"})
```

```
>>> b.update({"nombre":"Javier Perez"})
```

Organización y presentación de datos

```
>>> b.clear()
```

```
>>> del b["nombre"]
```

```
>>> b.pop("nombre")
```

```
>>> d=b.copy()
```

```
>>> b["edad"]=22
```

Diccionarios ordenados

```
>>> a=OrderedDict(nombre="Juan",rol="Admin")
```

Resumen

- Hemos visto en profundidad a los tipos de datos básicos que soporta Python
  - Datos numéricos y sus operadores relacionados
  - Las cadenas y hemos visto la indexación, el corte y el formato
  - Las listas (también conocidas como arrays)
  - Los booleanos y sus operadores
  - Diccionarios y sus tipos (dict; OrderedDict)
  - Los atributos definidos en los objetos diccionario