Python: Los básicos

Variables ampliadas por text (CONCATENATION)

Para encadenar texto

```
categoria1 = "verde"
color_detalle = categoria1 + ' ' + 'oscuro"

print(categoria1 + ' oscuro')
print(categoria1, 'oscuro')
```

type() and isinstance()

`float/int/str(variable)` cambia el tipo de data/type

`type(variable)` devuelve: class 'float/int/str'

`isinstance(variable, float/int/str)` comprobar el tipo de dato (devuelve True/False)

Operaciones Algebraicas

```
+ sumar              / dividir
- restar             // divider y redondear (modulus)
* multiplicar        % resto de una division (floor division)
** elevar            round(x) redondear número x
```

Operaciones Binarias

`==` comprobar si valores coinciden
`is` comprobar si valores son exacamente igual
`!=` comprobar si valores son diferentes
`is not` comprobar si valores no son exactamente iguales
`> (>=)` mayor que (mayor o igual que)
`< (<=)` menor que (menor o igual que)
and ambas verdaderas
or ambas o solo una verdadera
`in/not in` comprobar si hay un valor en una lista etc.

Metodos String

`string.upper()z` MAYUSCULAS
`string.lower()` minusculas
`string.capitalize()` Primera letra de la frase en may.
`string.title()` Primera Letra De Cada Palabra En May.
`string.swapcase()` mINUSCULAS A mAYUSCULAS O vICEVERSA
`string.strip()` quita espacios del principio y final

`string.split()` divide string en lista – por espacios por defecto, o especifica otro divisor en ()
`string.replace("frase", "frase")` remplaza la primera frase del string por el otro
`" ".join(string)` une los elementos de una lista en una string con el separador espificado en " "
`list(string)` convierte un variable string en una lista
`string.find("substring")` encuentra el indice en que empiece el substring/'-1' si no existe el substring

`string[i]` devuelve el elemento en la indice i
string[i:j] devuelve un rango de caracteres

Listas [ ] Metodos no permanentes

lista = [] crea una lista vacia

`len(lista)` devuelve el no. de elementos

`min(lista)/max(lista)` saca el valor minimo y maximo

`lista.count()` devuelve el no. de elementos que hay en la lista de un valor determinado en los()

`sorted(lista)` ordenar una lista de menor a mayor

`lista.copy()` hacer una copia de la lista

Metodos con indices

`list.index(x)` devuelve la indice de x en la lista
`lista[i]` devuelve el elemento en la indice i
[start:stop:step]
`lista[i:j:x]` devuelve los elementos por el rango de i a j (incluye i pero no j) saltando por x
`lista[-i:-j]` devuelve los elementos por los indices negativos (incluye –j pero no –i)

Listas – Acciones Permanentes

Ampliar una lista

[lista1, lista2] junta listas pero se mantienen como listas separadas
lista1 + lista2 hace una lista mas larga

.append()

`lista.append(x)#` añade un solo elemento (lista, string, integer o tuple) a la lista

.extend()

`lista.extend(lista2)#` añade los elementos de **una** lista al final de la lista

.insert()

`.insert(i, x)#` mete un elemento (x) en un índice(i)

Ordenar una lista

.sort()

`lista.sort()#` ordena de menor a mayor, usar con (reverse=True) para ordenar de mayor a menor
`lista.reverse()#` ordena los elementos al reves del orden guardado

Quitar elementos de una lista

.pop()

`lista.pop(i)#` quita el elemento en indice i y devuelve su valor

.remove()

`lista.remove(x)#` quita el primer elemento de la lista con valor x

`lista.clear()#` vacia la lista

`del lista#` borra la lista
`del lista[i]#` borra el elemento en indice i

Diccionarios { key : value , }

diccionario = {x:y} compuestos por un key(x) unica y un valor(y) (cualquier tipo de datos)

dict()

`variable = dict(x=y, m=n)` crear un diccionario

`dicc.copy()` crear una copia
`len(dicc)` devuelve el no. de elementos (x:y) hay en el diccionario
`sorted(dicc)` ordena los keys; usar con .items() para ordenar tuplas de los elementos o .values() para ordenar los values solos

Diccionarios – Metodos

Obtener informacion de un diccionario

`dicc.keys()` devuelve todas las keys
`dicc.values()` devuelve todos los values
`dicc.items()` devuelve tuplas de los key:value
in/not in comprobar si existe una clave
`dicc.get(x, y)` devuelve el valor asociado al key x, o si no existe devuelve el output y
`dicc["key"]` devuelve el valor del key (ver abajo que tiene mas usos)

Ampliar un diccionario

.update()

`dicc.update({x:y})#` para insertar nuevos elementos
`dicc["key"] = valor#` para inserter un nuevo key o valor, o cambiar el valor de un key
`dicc. setdefault(x, y)#` devuelve el value del key x, o si no existe la key x, la crea y asigna el valor y por defecto

Quitar elementos de un diccionario

`dicc.pop(x)#` elimina la key x (y lo devuelve)
`dicc.popitem()#` elimina el ultimo par de key:value
`dicc.clear()#` vacia el diccionario

Tuplas (,) inmutables, indexados

tupla = (x,y) tuplas se definen con () y , o solo ,
tupla1 + tupla2 juntar tuplas

tuple(lista) crear tuplas de una lista
`tuple(dicc)` crear tuplas de los keys de un diccionario
`tuple(dicc.values())` crear tuplas de los values
`tuple(dicc.items())` crear tuplas de los key:values

`len(tupla)` devuelve el no. de elementos
in/not in comprobar si hay un elemento
`tupla.index(x)` devuelve el indice de x
`tupla.count(x)` devuelve el no. de elementos con valor x en la tupla

para cambiar el contenido de una tupla hay que convertirla en una lista y luego a tupla

zip()

`zip(iterable1, iterable2)` crea una lista de tuplas de parejas de los elementos de las dos listas (mientras se puede)

`listzip.sort()` ordena las tuplas del zip por el primer elemento

Sets {}

no permiten duplicados, no tienen orden

set = {x,y}
`set(iterable)` solo permite un argumento iterable; elimina duplicados

in/not in comprobar si hay un elemento
`len(set)` devuelve el no. de elementos

Ampliar un set

`set.add(x)#` añadir un elemento
`set.update(set o lista)#` añadir uno o mas elementos con [] o {} o un variable tipo lista o set

Quitar elementos de un set

`set.pop()#` elimina un elemento al azar
`set.remove(x)#` elimina el elemento x
`set.discard(x)#` elimina el elemento x (y no devuelve error si no existe)
`set.clear()#` vacia el set

Operaciones con dos Sets

`set1.union(set2)` devuelve la union de los dos sets: todos los elementos menos dupl.
`set1.intersection(set2)` devuelve los elementos comunes de los dos sets
`set1.difference(set2)` devuelve los sets que estan en set1 pero no en set2 (restar)
`set1.symmetric_difference(set2)` devuelve todos los elementos que no estan en ambos
`set1.isdisjoint(set2)` comprobar si todos los elementos de dos sets son diferentes
`set1.issubset(set2)` comprobar si todos los elementos de set1 estan en set2
`set1.superset(set2)` comprobar si todos los elementos de set2 estan en set1

input()

- permite obtener texto escrito por teclado del usuario

`input("el texto que quieres mostrar al usuario")`

- se puede guardar en un variable
- por defecto se guarda como un string

`x = int(input("escribe un número")` para usar el variable como integer o float se puede convertir en el variable

Sentencias de control

`if ... elif ... else`
`if` estableca una condición para que se ejecute el código que esta debajo del if. *tiene que estar indentado*
`elif` para chequear mas condiciones después de un if
`else` agrupa las condiciones que no se han cumplido; no puede llevar condiciones nuevas

```
if x > y:
    print("x es mayor que y")
elif x == y:
    print("x es igual que y")
else:
    print("x e y son iguales")
```

`while`
- repite el código mientras la condición sea True, o sea se parará cuando la condición sea False
- se pueden incluir condiciones con if... elif... else
- *pueden ser infinitos* (si la condición no llega a ser False)

```
while x < 5:
    print("x es mayor que 5")
```

For loops

- sirven para iterar por todos los elementos de un variable que tiene que ser un iterable (lista, diccionario, tupla, set, or string)
- se pueden combinar con if ... elif ... else, while, u otro for loop
- en diccionarios por defecto intera por las keys; podemos usar dicc.values() para acceder a los values

```
for i in lista:
    print("hola mundo")
```

List comprehension

- su principal uso es para crear una lista nueva de un un for loop en una sola línea de codigo

[lo que queremos obtener `iterable` `condición` (opcional)]

try ... except

Se usan para evitar que nuestro código se pare debido a un error en el código. Se puede imprimir un mensaje que avisa del error.

```
try:
    print("2.split())
except:
    print("no funciona")
```

range()

- nos devuelve una lista de números que por defecto se aumentan de uno en uno empezando por 0

`range(start:stop:step)`

- se puede especificar por donde empieza y el limite (que debe ser +1 por que se para uno antes del limite que ponemos como stop)
- tambien se puede especificar saltos

 # metodos permanentes (cambia el variable, no devuelve nada)