Guía de Referencia Rápida de Python

Generalidades

- Una declaración debe estar toda en una sola línea. Para romper una declaración en múltiples líneas debe usarse '\' al final de cada una de ellas (salvo la última).

  Excepción: siempre se puede romper dentro de cualquier par `( )`, `[ ]` o `{ }`, o en una cadena delimitada por comillas triples.

- En una línea pueden aparecer más de una declaración separándolas por ';'.
- Los comentarios comienzan con '#' y continúan hasta el final de la línea.
- Un identificador está formado por una letra o símbolo '_' seguido de más letras, números o símbolos '_'.
- Python distingue mayúsculas de minúsculas.

Tipos de datos

Los tipos de datos en Python se dividen en mutables, si su contenido puede cambiarse, e inmutables, si su contenido no puede cambiarse.

Constantes lógicas (*bool*)

- `True` (verdadero).
- `False` (falso).
- Las constantes lógicas son datos inmutables.

Números

- Enteros (*int*): `1234`, `-123456789`.
- Reales o números en coma flotante (*float*): `0.001`, `10.`, `3.14e-10`.
- Complejos (*complex*): `2+3j`, `.5-1.4j`, `1j`.
- Todos los tipos de números son datos inmutables.

Secuencias

- Listas (*list*): delimitadas por corchetes y los elementos separados por comas: `[ ]`, `[1]`, `[1, 2, 3]`.
- Tuplas (*tuple*): delimitadas por paréntesis y los elementos separados por comas: `( )`, `(1,)`, `(1, 2, 3)`.
- Rangos (*range*): se crean con la función `range`: `range(10)`, `range(5, 10)`, `range(2, 10, 2)`.
- Cadenas (*string*): delimitadas por comillas simples (`'Hola'`), dobles (`"Hola"`) o triples (`'''Hola'''`, `"""Hola"""`).
- Las listas son datos mutables. Las tuplas, rangos y cadenas son datos inmutables.

Diccionarios (*dict*)

- Pares `clave:valor` separados por coma y delimitados por llaves: `{1:'primero', 'segundo':2}`.
- Las claves deben ser datos inmutables.
- Los diccionarios son datos mutables.

Conjuntos

- Los conjuntos son colecciones no ordenadas de elementos no duplicados.
- Los elementos de un conjunto deben ser datos inmutables.
- `set(secuencia)` crea un conjunto mutable con los elementos de la secuencia especificada, descartando las repeticiones. Salvo para el conjunto vacío, se puede construir un conjunto encerrando sus elementos entre llaves: `{1, 2, 3}`.
- `frozenset(secuencia)` crea un conjunto inmutable con los elementos de la secuencia especificada, descartando las repeticiones.

Otros tipos de datos

- La constante `None` es el dato nulo.

Operaciones sobre los tipos de datos básicos

Operadores lógicos

Declaración	*Evalúa a*
`bool(expr)`	`True` si `expr` es verdadera, `False` en caso contrario
`expr1 or expr2`	`False` si `expr1` y `expr2` son falsos, `True` en caso contrario
`expr1 and expr2`	`True` si `expr1` y `expr2` son verdaderos, `True` en caso contrario
`not expr`	`True` si `expr` es falsa, `False` en caso contrario

- La constante `None`, los ceros numéricos, las secuencias vacías y los diccionarios y conjuntos vacíos se consideran falsos. El resto de datos se consideran verdaderos.
- Los operadores **`or`** y **`and`** solo evalúan `expr2` en caso necesario.

Operadores de comparación

Operador	*Significado*
`<`	Menor estricto que
`<=`	Menor o igual que
`>`	Mayor estricto que
`>=`	Mayor o igual que
`==`	Igual que
`!=`	Distinto que
`is`	Idéntico a
`is not`	No idéntico a

- Están definidos entre cualesquiera tipos de datos.

Operadores numéricos

Operación	*Resultado*
`x+y`	Suma de `x` e `y`
`x-y`	Diferencia de `x` e `y`

Operación	*Resultado*
`x*y`	Producto de `x` e `y`
`x/y`	División de `x` por `y`
`x**y`	`x` elevado a `y`
`x//y`	Cociente de la división de `x` por `y`
`x%y`	Resto de la división de `x` por `y`
`divmod(x,y)`	La tupla `(x//y, x%y)`
`-x`	`x` negado
`abs(x)`	Valor absoluto de `x`
`round(x, n)`	Redondea `x` al valor más cercano con `n` dígitos tras la coma decimal. El valor por defecto de `n` es `0`
`int(x)`	`x` convertido a entero
`float(x)`	`x` convertido a real

Operaciones sobre secuencias

Operación	*Resultado*
`x in s`	`True` si un elemento de `s` es igual a `x`, `False` en caso contrario
`x not in s`	`False` si un elemento de `s` es igual a `x`, `True` en caso contrario
`s1+s2`	Concatenación de `s1` y `s2`
`s*n` `n*s`	`n` copias de `s` concatenadas
`s[i]`	`i`-ésimo elemento de `s`
`s[i:j:salto]`	Porción de `s` con los elementos en los índices de la progresión aritmética desde `i` (incluido) hasta `j` (excluido) con diferencia `salto`
`len(s)`	El número de elementos de `s`
`max(s)`	Mayor elemento de `s`
`min(s)`	Menor elemento de `s`
`sorted(s,clave,inv)`	Lista con los elementos de `s` en orden. `clave` es una función que proporciona de cada elemento de `s` su clave de comparación. `inv` determina si el orden es ascendente o descendente
`sum(s, valini)`	Devuelve la suma de la secuencia de números `s`, añadiéndole además `valini`. El valor por defecto de `valini` es `0`

- Si `s` es una cadena, entonces los operadores `in` y `not in` comprueban si `x` es una subcadena o no es una subcadena de `s`.
- Si `i` o `j` son negativos, entonces son relativos al final de la secuencia; es decir, se considera `len(s)+i` o `len(s)+j` en su lugar.
- Si `i` o `j` son mayores que `len(s)`, entonces se considera `len(s)` en su lugar.
- El valor por defecto de `j` es `len(s)`. El valor por defecto de `salto` es `1`.

Operaciones sobre cadenas

Operación	*Resultado*
`s.isalnum()`	`True` si todos los caracteres de `s` son alfanuméricos, `False` en caso contrario
`s.isalpha()`	`True` si todos los caracteres de `s` son alfabéticos, `False` en caso contrario
`s.isdigit()`	`True` si todos los caracteres de `s` son dígitos, `False` en caso contrario
`s.islower()`	`True` si todos los caracteres de `s` están en minúsculas, `False` en caso contrario
`s.isspace()`	`True` si todos los caracteres de `s` son espacios en blanco, `False` en caso contrario
`s.isupper()`	`True` si todos los caracteres de `s` están en mayúsculas, `False` en caso contrario
`s.capitalize()`	Una copia de `s` con solo el primer carácter en mayúsculas
`s.lower()`	Una copia de `s` con todos los caracteres de `s` en minúsculas
`s.upper()`	Una copia de `s` con todos los caracteres en mayúsculas
`s.swapcase()`	Una copia de `s` transformando mayúsculas en minúsculas y viceversa
`s.count(sub,ini,fin)`	Número de ocurrencias de la subcadena `sub` en `s[ini:fin]`
`s.find(sub,ini,fin)`	El menor índice donde se encuentra `sub` como subcadena de `s[ini:fin]`, `-1` si no se encuentra

Operación	*Resultado*
`s.index(sub,ini,fin)`	Igual que `s.find(sub,ini,fin)`, pero provoca un error si la subcadena no se encuentra
`s.rfind(sub,ini,fin)`	El mayor índice donde se encuentra `sub` como subcadena de `s[ini:fin]`, `-1` si no se encuentra
`s.rindex(sub,ini,fin)`	Igual que `s.rfind(sub,ini,fin)`, pero provoca un error si la subcadena no se encuentra
`s.join(sec)`	Concatenación de las cadenas contenidas en `sec`, separadas por `s`
`s.replace(vie, nue, max)`	Una copia de `s` con todas (o `max`) las ocurrencias de la subcadena `vie` reemplazadas por la subcadena `nue`
`s.split(sep,max)`	La lista de todas (o `max`) palabras obtenidas de `s` usando `sep` como cadena delimitadora
`s.rsplit(sep,max)`	Igual que `s.split(sep,max)`, pero desde el final de la cadena
`s.format(args)`	Realiza una operación de formateo. La cadena `s` puede contener texto literal junto con campos de reemplazamiento delimitados por llaves.

- El valor por defecto de `ini` es `0`, y el valor por defecto de `fin` es `len(s)`.
- Los métodos `isalnum`, `isalpha`, `isdigit` e `isspace` devuelven `False` si la cadena no contiene al menos un carácter. Los métodos `islower` e `isupper` devuelven `False` si la cadena no contiene al menos un carácter alfabético.
- El valor por defecto de `sep` es espacio en blanco. El argumento `max` es opcional.
- Ejemplos de operaciones de formateo:
  1. `'{}, {}, {}'.format('a', 'b', 'c')` proporciona `'a, b, c'`
  2. `'{2}, {1}, {0}'.format('a', 'b', 'c')` proporciona `'c, b, a'`
  3. `'{0}{1}{0}'.format('abra', 'cad')` proporciona `'abracadabra'`
  4. `'Valor de PI: {parteentera},{partedecimal}'.format( parteentera=3, partedecimal=1415)` proporciona `'Valor de PI: 3,1415'`

Operaciones sobre listas

Operación	*Resultado*
`s[i]=x`	El `i`-ésimo elemento de `s` es reemplazado por `x`
`del s[i]`	Elimina el `i`-ésimo elemento de `s`
`s[i:j]=t`	La porción de `s` con los elementos en las posiciones de la progresión aritmética desde `i` (incluido) hasta `j` (excluido) es reemplazada por `t`
`del s[i:j]`	Lo mismo que `s[i:j]=[]`
`s[i:j:salto]=t`	Los elementos de `s` en las posiciones de la progresión aritmética desde `i` (incluido) hasta `j` (excluido) con diferencia `salto` son reemplazados por los elementos de `t`
`del s[i:j:salto]`	Los elementos de `s` en las posiciones de la progresión aritmética desde `i` (incluido) hasta `j` (excluido) con diferencia `salto` son eliminados
`s.append(x)`	Añade `x` al final de `s`
`s.extend(x)`	Añade los elementos de `x` al final de `s`
`s.count(x)`	El cardinal del conjunto `{i : s[i]==x}`
`s.index(x,ini,fin)`	El menor `i` tal que `ini` $\leq$ `i` $<$ `fin` y `s[i]==x`
`s.insert(i,x)`	Inserta `x` en la posición `i`-ésima de `s`
`s.remove(x)`	Elimina la primera ocurrencia de `x` en `s`
`s.pop(i)`	Lo mismo que `x=s[i]; del s[i]; return x`
`s.reverse()`	Invierte el orden de los elementos de `s`
`s.sort(clave, inv)`	Ordena los elementos de `s`

- En la operación `s[i:j:salto] = t`, la secuencia `t` debe tener la misma longitud que la secuencia `s`.
- Los métodos `reverse` y `sort` modifican la lista. No devuelven el resultado para poner de manifiesto este efecto lateral.
- El argumento del método `pop` toma `-1` como valor por defecto, por lo que por defecto el último elemento de la lista es eliminado y devuelto.

Operaciones sobre diccionarios

Operación	*Resultado*
`d[k]`	El elemento de `d` con clave `k`
`d[k]=x`	Establece a `x` el valor del elemento de `d` con clave `k`
`del d[k]`	Elimina el elemento de `d` con clave `k`
`k in d`	`True` si `k` es una clave de `d`, `False` en caso contrario
`k not in d`	`True` si `k` no es una clave de `d`, `False` en caso contrario
`d.clear()`	Elimina todos los elementos de `d`
`d.copy()`	Proporciona una copia de `d`
`d.get(k, x)`	Si `k` es una clave de `d` devuelve su valor, en caso contrario devuelve `x`
`d.setdefault(k, x)`	Si `k` es una clave de `d` devuelve su valor, en caso contrario inserta en `d` la clave `k` con valor `x`
`d.items()`	Los pares `(clave, valor)` de `d`
`d.keys()`	Las claves de `d`
`d.values()`	Los valores de `d`

Operaciones sobre conjuntos

Operación	*Resultado*
`elt in s`	`True` si `elt` pertenece a `s`, `False` en caso contrario
`elt not in s`	`True` si `elt` no pertenece a `s`, `False` en caso contrario
`s1.isdisjoint(s2)`	`True` si `s1` no tiene ningún elemento en común con `s2`, `False` en caso contrario
`s1.issubset(s2)`	`True` si todo elemento de `s1` pertenece a `s2`, `False` en caso contrario
`s1.issuperset(s2)`	`True` si todo elemento de `s2` pertenece a `s1`, `False` en caso contrario
`s1.intersection(s2)` `s1 & s2`	Nuevo conjunto con los elementos comunes a `s1` y `s2`
`s1.union(s2)` `s1 | s2`	Nuevo conjunto con los elementos de `s1` y de `s2`
`s1.difference(s2)` `s1 - s2`	Nuevo conjunto con los elementos que pertenecen a `s1`, pero no a `s2`

Operación	*Resultado*
`s1.symmetric_difference(s2)` `s1 ^ s2`	Nuevo conjunto con los elementos que pertenecen a `s1` o a `s2`, pero no a ambos
`s.copy()`	Nuevo conjunto copia del conjunto `s`
`s.add(elt)`	Añade `elt` al conjunto mutable `s`
`s.update(sec)`	Añade los elementos de la secuencia `sec` al conjunto mutable `s`
`s.discard(elt)`	Elimina `elt` del conjunto mutable `s`
`s.remove(elt)`	Elimina `elt` del conjunto mutable `s`, pero provoca un error si `elt` no está en `s`
`s.clear()`	Elimina todos los elementos del conjunto mutable `s`

Declaraciones

Declaración	*Resultado*
`pass`	Declaración nula
`del nombre`	Borra el dato llamado `nombre`
`global nombre`	Establece como global la variable `nombre`
`raise TipoError`	Provoca un error de tipo `TipoError`

Operadores de asignación

Operador	*Resultado*
`a=b`	Asigna el dato `b` a la etiqueta `a`
`a+=b`	Lo mismo que `a=a+b`
`a-=b`	Lo mismo que `a=a-b`
`a*=b`	Lo mismo que `a=a*b`
`a/=b`	Lo mismo que `a=a/b`
`a%=b`	Lo mismo que `a=a%b`
`a**=b`	Lo mismo que `a=a**b`

- El operador de asignación puede desempaquetar secuencias:

  `(a,b)=range(2)` es lo mismo que `a=0;b=1`
  `x,y=y,x` intercambia los valores de `x` e `y`

- Es posible realizar asignaciones múltiples:

  `a=b=c=0` es lo mismo que `a=0; b=0; c=0`

Declaraciones de control de flujo

Declaración	*Significado*

`if condición:`
`  consecuencias`
`elif condición:`
`  consecuencias`
`else:`
`  alternativas`
	Condicionales simple, doble y múltiple

`while condición:` `  acciones`	Bucle mientras
`for elt in secuencia:` `  acciones`	Bucles para y desde
`break`	Interrupción de un bucle
`continue`	Continuación de un bucle

Definición de funciones

```
def nombre_función (parámetros):
  'documentación'
  acciones
  return resultado
```

- `parámetros` es una sucesión de identificadores separados por comas:

  def nula (x, y, z):
  'Función que no hace nada'
  pass

- Los argumentos se pasan a la función por posición o por nombre:

  ```
  nula(1, z=3, 2)
  ```

  hace que los parámetros tomen los siguientes valores:

  ```
  x=1, y=2, z=3
  ```

- La sucesión de parámetros puede contener uno de la forma `*nombre`, en cuyo caso se le asignará la tupla de todos los argumentos proporcionados por posición que no correspondan a otro parámetro:

```
def nula (x, y, z, *args):
    'Función que no hace nada'
    pass
nula(1, 2, 3, 4, 5, 6)
```

hace que los parámetros tomen los siguientes valores:

```
x=1, y=2, z=3, args=(4, 5, 6)
```

- La sucesión de parámetros puede contener uno de la forma `**nombre`, en cuyo caso se le asignará un diccionario con todos los argumentos proporcionados por nombre que no correspondan a otro parámetro:

```
def nula (x, y, z, **kwargs):
    'Función que no hace nada'
    pass
nula(1, z=3, 2, u=4, v=5, w=6)
```

hace que los parámetros tomen los siguientes valores:

```
x=1, y=2, z=3, kwargs={u:4, v:5, w:6}
```

- La declaración **`return`** devuelve el resultado de aplicar la función a los argumentos proporcionados. Si no se incluye, entonces la función devuelve `None` (y, entonces, la consideramos un procedimiento).

Funciones predefinidas

Función	*Resultado*
`help()`	Invoca el sistema de ayuda
`dir()`	Devuelve la lista de variables definidas
`vars()`	Devuelve un diccionario con los nombres y valores de las variables definidas
`globals()`	Devuelve un diccionario con los nombres y valores de las variables globales definidas
`locals()`	Devuelve un diccionario con los nombres y valores de las variables locales definidas
`eval(cadena)`	Evalúa `cadena` como una expresión de Python

Función	*Resultado*
`input(mensaje)`	Escribe `mensaje` en pantalla, lee una línea desde el teclado y la devuelve como una cadena
`isinstance(dato, tipodato)`	Comprueba si `dato` es del tipo especificado
`str(dato)`	Devuelve una cadena conteniendo una representación escribible de `dato`
`print(`s_1`, ..., `s_n`, sep=' ', end='\n')`	Escribe en pantalla representaciones de s_1, ..., s_n separadas por `sep` y seguidas por `end`

Definición de clases de objetos

```
class nombre_clase:
  definición de métodos
```

define la clase básica `nombre_clase` y

```
class nombre_clase (nombre_superclase):
  definición de métodos
```

define la clase `nombre_clase` que hereda de la clase `nombre_superclase`.

Métodos especiales y redefinición de operadores

Método	*Descripción*
`__init__(propio,args)`	Procedimiento que inicializa la instancia a partir de `args`
`__str__(propio)`	Función que devuelve una cadena representando la instancia
`__lt__(propio, otro)`	Función utilizada para la comparación `propio<otro`
`__le__(propio, otro)`	Función utilizada para la comparación `propio<=otro`
`__gt__(propio, otro)`	Función utilizada para la comparación `propio>otro`
`__ge__(propio, otro)`	Función utilizada para la comparación `propio>=otro`

Método	*Descripción*
`__eq__(propio, otro)`	Función utilizada para la comparación `propio==otro`
`__ne__(propio, otro)`	Función utilizada para la comparación `propio!=otro`
`__add__(propio, otro)`	Función utilizada para la operación `propio+otro`
`__sub__(propio, otro)`	Función utilizada para la operación `propio-otro`
`__mul__(propio, otro)`	Función utilizada para la operación `propio*otro`
`__truediv__(propio, otro)`	Función utilizada para la operación `propio/otro`
`__floordiv__(propio, otro)`	Función utilizada para la operación `propio//otro`
`__mod__(propio, otro)`	Función utilizada para la operación `propio%otro`
`__pow__(propio, otro)`	Función utilizada para la operación `propio**otro`
`__neg__(propio)`	Función utilizada para la operación `-propio`
`__len__(propio)`	Función utilizada para la operación `len(propio)`
`__getitem__(propio, k)`	Función utilizada para la operación `propio[k]`
`__setitem__(propio, k, valor)`	Función utilizada para la operación `propio[k]=valor`
`__delitem__(propio, k)`	Función utilizada para la operación **`del`** `propio[k]`
`__contains__(propio, elt)`	Función utilizada para la operación `elt` **`in`** `propio`

Módulos

Declaración	*Resultado*
`import` módulo$_1$, ..., módulo$_n$	Importa los módulos especificados
`from` módulo `import` nombre$_1$, ..., nombre$_n$	Importa los nombres especificados del módulo indicado
`from` módulo `import *`	Importa todos los nombres definidos en el módulo especificado

Módulo string

Constante	*Valor*
`digits`	La cadena `'0123456789'`
`ascii_letters`	La cadena con todos los caracteres alfabéticos (salvo la ñ) en minúsculas y en mayúsculas
`ascii_lowercase`	La cadena con todos los caracteres alfabéticos (salvo la ñ) en minúsculas
`ascii_uppercase`	La cadena con todos los caracteres alfabéticos (salvo la ñ) en mayúsculas

Módulo math

Constante	*Valor*
`pi`	`3.141592...`
`e`	`2.718281...`

Función	*Resultado*
`acos(x)`	El arco-coseno de `x`, en radianes
`asin(x)`	El arco-seno de `x`, en radianes
`atan(x)`	El arco-tangente de `x`, en radianes
`atan2(y, x)`	El arco-tangente de `y/x`, en radianes. El resultado está entre `-pi` y `pi`, y se consideran los signos de `x` e `y`, al contrario que `atan(y/x)`
`ceil(x)`	El menor entero mayor o igual que `x`
`cos(x)`	El coseno del ángulo en radianes `x`
`degrees(x)`	Convierte el ángulo `x` de radianes a grados
`exp(x)`	`e**x`
`fabs(x)`	Valor absoluto de `x` como un número real

Función	*Resultado*
`factorial(x)`	Devuelve el factorial de `x`
`floor(x)`	El mayor entero menor o igual que `x`
`hypot(x,y)`	La norma Euclídea `sqrt(x*x+y*y)`
`log(x, base)`	Logaritmo de `x` en la base especificada (por defecto `e`)
`log10(x)`	Logaritmo en base `10` de `x`
`modf(x)`	Tupla con la parte decimal y la parte entera (como número real) de `x`. Ambos números conservan el signo de `x`
`radians(x)`	Convierte el ángulo `x` de grados a radianes
`sin(x)`	El seno del ángulo en radianes `x`
`sqrt(x)`	La raíz cuadrada de `x`
`tan(x)`	La tangente del ángulo en radianes `x`