**Funciones principales en Turtle Graphics**

<https://www.programoergosum.com/cursos-online/raspberry-pi/245-turtle-graphics-en-python-con-raspberry-pi/mi-primera-figura>

Las funciones principales para animar nuestro objeto son las siguientes:

**forward(distance)**: Avanzar una determinada cantidad de píxeles.  
**backward(distance)**: Retroceder una determinada cantidad de píxeles.  
**left(angle)**: Girar hacia la izquierda un determinado ángulo.  
**right(angle)**: Girar hacia la derecha un determinado ángulo.

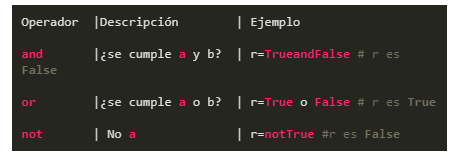
Por otro lado, puede que en ocasiones queramos desplazarnos de un punto a otro sin dejar rastro. Para ello utilizaremos las siguientes funciones:

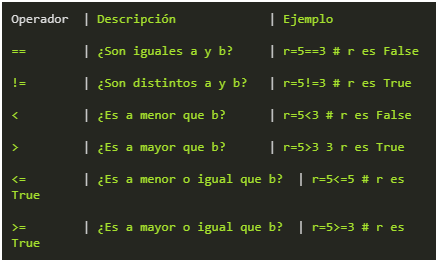
**home(distance)**: Desplazarse al origen de coordenadas.  
**goto((x, y))**: Desplazarse a una coordenada en concreto.  
**pendown()**: Subir el lápiz para no mostrar el rastro.  
**penup()**: Bajar el lápiz para mostrar el rastro.

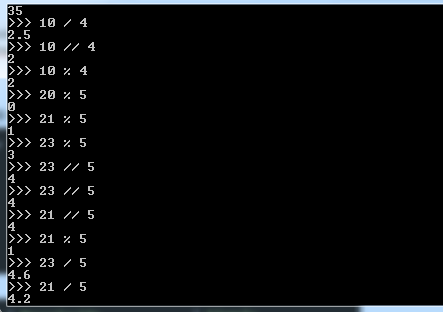
Por último, puede que queramos cambiar el color o tamaño del lápiz. En ese caso utilizaremos las siguientes funciones gráficas:

**shape(‘turtle’)**: Cambia al objeto tortuga.  
**pencolor(color)**: Cambiar al color especificado.  
**pensize(dimension)**: Tamaño de la punta del lápiz.

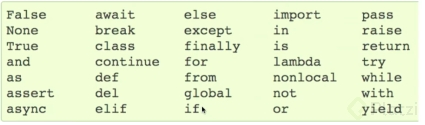
* **OPERADOS MATEMATICOS**







* **PALABRAS RESERVADAS**



* **FUNCIONES IMPORTANTES**

<https://docs.python.org/3/library/functions.html#any>

**Operaciones con Strings en Python**

Los strings tienen varios métodos que nosotros podemos utilizar.

* **upper**: convierte todo el **string** a mayúsculas
* **lower:** convierte todo el **string** a minúsculas
* **find:** encuentra el indice en donde existe un patrón que nosotros definimos
* **startswith:** significa que empieza con algún patrón.
* **endswith:** significa que termina con algún patrón
* **capitalize:** coloca la primera letra en mayúscula y el resto en minúscula
* **in y not in** nos permite saber con cualquier secuencia sin una

subsecuencia o substrings se encuentra adentro de la secuencia mayor.

* **dir:** Nos dice todos los métodos que podemos utilizar dentro de un objeto.
* **help:** nos imprime en pantalla el *docstrings* o comentario de ayuda o instrucciones que posee la función. Casi todas las funciones en Python las tienen.

Ej:



Para limpiar en el cmd de windows usa:

>>> import os

>>> clear = lambda: os.system('cls')

>>> clear()

y despues ya puedes usar solo

>>> clear()

* **METODOS STRING**

|  |  |
| --- | --- |
| * help(str) dir(str) help(str.splitlines) import time help(time)   capitalize() casefold() center(width[, fillchar]) count(sub[, start[, end]]) encode(encoding=“utf-8”, errors=“strict”) endswith(suffix[, start[, end]]) expandtabs(tabsize=8) find(sub[, start[, end]]) format(\*args, \*\*kwargs) format\_map(mapping) index(sub[, start[, end]]) isalnum() isalpha() isdecimal() isdigit() isidentifier() islower() translate(table) upper() zfill(width) | isnumeric() isprintable() isspace() istitle() isupper() join(iterable) ljust(width[, fillchar]) lower() lstrip([chars]) maketrans(x[, y[, z]]) partition(sep) replace(old, new[, count]) rfind(sub[, start[, end]]) rindex(sub[, start[, end]]) rjust(width[, fillchar]) rpartition(sep) rsplit(sep=None, maxsplit=-1) rstrip([chars]) split(sep=None, maxsplit=-1) splitlines([keepends]) startswith(prefix[, start[, end]]) strip([chars]) swapcase() title() |

**CRUD**

(C)reate

(R)ead or retreive

(U)pdate

(D)elete

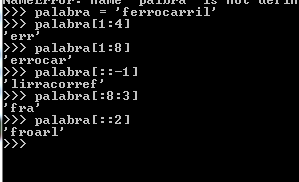
**Operaciones con strings: Slices en python**

Los *slices* en Python nos permiten manejar secuencia de una manera poderosa.

**Slices** en español significa ““rebanada””, si tenemos una secuencia de elementos y queremos una rebanada tenemos una sintaxis para definir que pedazos queremos de esa secuencia.

secuencia[comienzo:final:pasos]

EJ:



**For loops**

Las iteraciones es uno de los conceptos **más importantes** en la programación. En **Python** existen muchas manera de iterar pero las dos principales son los **for loops** y **while loops**.

Los **for loops** nos permiten iterar a través de una secuencia y los while loops nos permiten iterara hasta cuando una condición se vuelva falsa.

**for loops**:

* Tienen dos keywords break y continue que nos permiten salir anticipadamente de la iteración
* Se usan cuando se quiere ejecutar varias veces una o varias instrucciones.
* for [variable] in [secuencia]:

Es una convención usar la letra i como variable en nuestro for, pero podemos colocar la que queramos.

range: Nos da un objeto rango, es un iterador sobre el cual podemos generar secuencias

**While loops**

Al igual que las for loops, las **while loops** nos sirve para iterar, pero las for loops nos sirve para iterar a lo largo de una secuencia mientras que las **while loops** nos sirve para iterar mientras una condición sea verdadera.

Si no tenemos un mecanismo para convertir el mecanismo en falsedad, entonces nuestro while loops se ira al infinito(infinite loop).

**Uso de listas**

Python y todos los lenguajes nos ofrecen *constructos* mucho más poderoso, haciendo que el desarrollo de nuestro software sea

* Más sofisticado
* Más legible
* Más fácil de implementar

Estos *constructos* se llaman **Estructuras de Datos** que nos permiten agrupar de distintas manera varios valores y elementos para poderlos manipular con mayor facilidad.

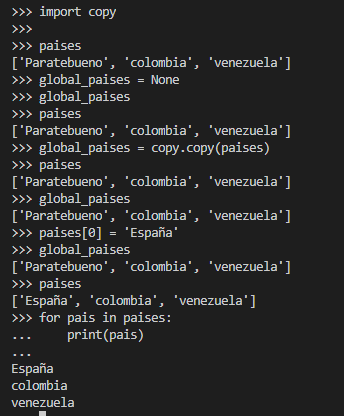
Las **listas** las vas a utilizar durante toda tu carrera dentro de la programación e ingeniería de Software.

Las **listas** son una secuencia de valores. A diferencia de los *strings*, las **listas** pueden tener cualquier tipo de valor. También, a diferencia de los strings, son mutables, podemos agregar y eliminar elementos.

En Python, las listas son referenciales. Una lista no guarda en memoria los objetos, sólo guarda la **referencia** hacia donde viven los objetos en memoria

Se inician con [] o con la *built-in function* list.

**Uso de import copy:**



**Operaciones con listas**

Ahora que ya entiendes cómo funcionan las **listas**, podemos ver qué tipo de operaciones y métodos podemos utilizar para modificarlas, manipularlas y realizar diferentes tipos de cómputos con esta Estructura de Datos.

* El operador **+(suma)** concatena dos o más listas.
* El operador \***(multiplicación)** repite los elementos de la misma lista tantas veces los queramos multiplicar

Sólo podemos utilizar **+(suma)** y \***(multiplicación)**.

Las listas tienen varios métodos que podemos utilizar.

* **append** nos permite añadir elementos a listas. Cambia el tamaño de la lista.
* **pop** nos permite sacar el último elemento de la lista. También recibe un índice y esto nos permite elegir qué elemento queremos eliminar.
* **sort** modifica la propia lista y ordenarla de mayor a menor. Existe otro método llamado **sorted,** que también ordena la lista, pero genera una nueva instancia de la lista
* **del** nos permite eliminar elementos vía indices, funciona con *slices*
* **remove** nos permite es pasarle un valor para que Python compare internamente los valores y determina cuál de ellos hace match o son iguales para eliminarlos.

Ej:

**>>> a = list(range(0, 100, 2))**

**>>> a**

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98]

**>>> b= list(range(0, 10, 2))**

**>>> a + b**

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 0, 2, 4, 6, 8]

**>>> b \* 2**

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

**>>> a/b**

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for /: 'list' and 'list'

**>>> a % b**

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

**>>> a – b**

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

**>>> fruits = list()**

**>>> fruits**

[]

>>> **fruits.append('apple')**

**>>> fruits**

['apple']

**>>> fruits.append('banana')**

**>>> fruits**

['apple', 'banana']

**>>> len(fruits)**

2

**>>> fruits.append('kiwi')**

**>>> len(fruits)**

3

**>>> fruits**

['apple', 'banana', 'kiwi']

**>>> some\_fruit = fruits.pop()**

**>>> some\_fruit**

'kiwi'

**>>> fruits**

['apple', 'banana']

**>>> some\_fruit = fruits.pop(0)**

**>>> some\_fruit**

'apple'

**>>> fruits**

['banana']

**>>> del fruits[0]**

**>>> fruits**

[] borro el ultimo de la lista

>>>

**import random**

**>>>**

**>>> random\_numbers = []**

**>>> for i in range(10): random\_numbers.append(random.randint(0, 15))**

**>>> random\_numbers**

[4, 8, 7, 9, 0, 2, 15, 8, 3, 12]

**>>> ordenar = sorted(random\_numbers)**

**>>> random\_numbers**

[4, 8, 7, 9, 0, 2, 15, 8, 3, 12]

**>>> ordenar**

[0, 2, 3, 4, 7, 8, 8, 9, 12, 15]

**>>> random\_numbers.sort()**

**>>> random\_numbers**

[0, 2, 3, 4, 7, 8, 8, 9, 12, 15]

***VER TODOS LOS METODOS DE LA LIST***

**>>> dir(random\_numbers)**

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

list.clear()

>>> my\_list = list(range(20))

my\_list.insert(2, 30)

print("new list:", my\_list)

newlist: [0, 1, 30, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

**Diccionarios**

Los diccionarios se conocen con diferentes nombres a lo largo de los lenguajes de programación como HashMaps, Mapas, Objetos, etc. En Python se conocen como **Diccionarios**.

Un diccionario es similar a una lista sabiendo que podemos acceder a través de un indice, pero en el caso de las listas este índice debe ser un número entero. Con los diccionarios puede ser cualquier objeto, normalmente los verán con **strings** para ser más explicitos, pero funcionan con muchos tipos de llaves…

Un diccionario es una asociación entre llaves(**keys**) y valores(**values**) y la referencia en Python es muy precisa. Si abres un diccionario verás muchas palabras y cada palabra tiene su definición.

Para iniciar un diccionario se usa {} o con la función dict

Estos también tienen varios métodos. Siempre puedes usar la función dir para saber todos los métodos que puedes usar con un objeto.

Si queremos ciclar a lo largo de un diccionario tenemos las opciones:

**keys**: nos imprime una lista de las llaves  
**values** nos imprime una lista de los valores  
**items**. nos manda una lista de tuplas de los valores

EJ :

rae = {}

>>> rae['pizza'] = 'la comida mas rica del mundo'

>>> rae

{'pizza': 'la comida mas rica del mundo'}

>>> rae['pasta'] = 'la comida mas sabrosa de italia'

>>> rae

{'pizza': 'la comida mas rica del mundo', 'pasta': 'la comida mas sabrosa de italia'}

>>> rae['pizza']

'la comida mas rica del mundo'

>>> rae['pasta']

'la comida mas sabrosa de italia'

***ERROR:***

>>>>>> a = rae.get('helado', None)

>>> print(a) = None

>>> a = rae.get('pizza', None)

>>> print(a) = la comida mas rica del mundo

>>> rae.keys() = dict\_keys(['pizza', 'pasta'])

>>> rae.values()

dict\_values(['la comida mas rica del mundo', 'la comida mas sabrosa de italia'])

>>> rae.items()

dict\_items([('pizza', 'la comida mas rica del mundo'), ('pasta', 'la comida mas sabrosa de italia')]) se hace una tupla

>>> for key in rae.keys():

... print(key)

pizza

pasta

>>> for key in rae.values():

... print(key)

la comida mas rica del mundo

la comida mas sabrosa de Italia

**>>> for key, value in rae.items():**

... print(key, value)...

pizza la comida mas rica del mundo

pasta la comida mas sabrosa de Italia

**todas las funcions:**

>>> dir(rae)

['\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']