

Fundamentos de programación

Alex Di Genova

16/04/2024

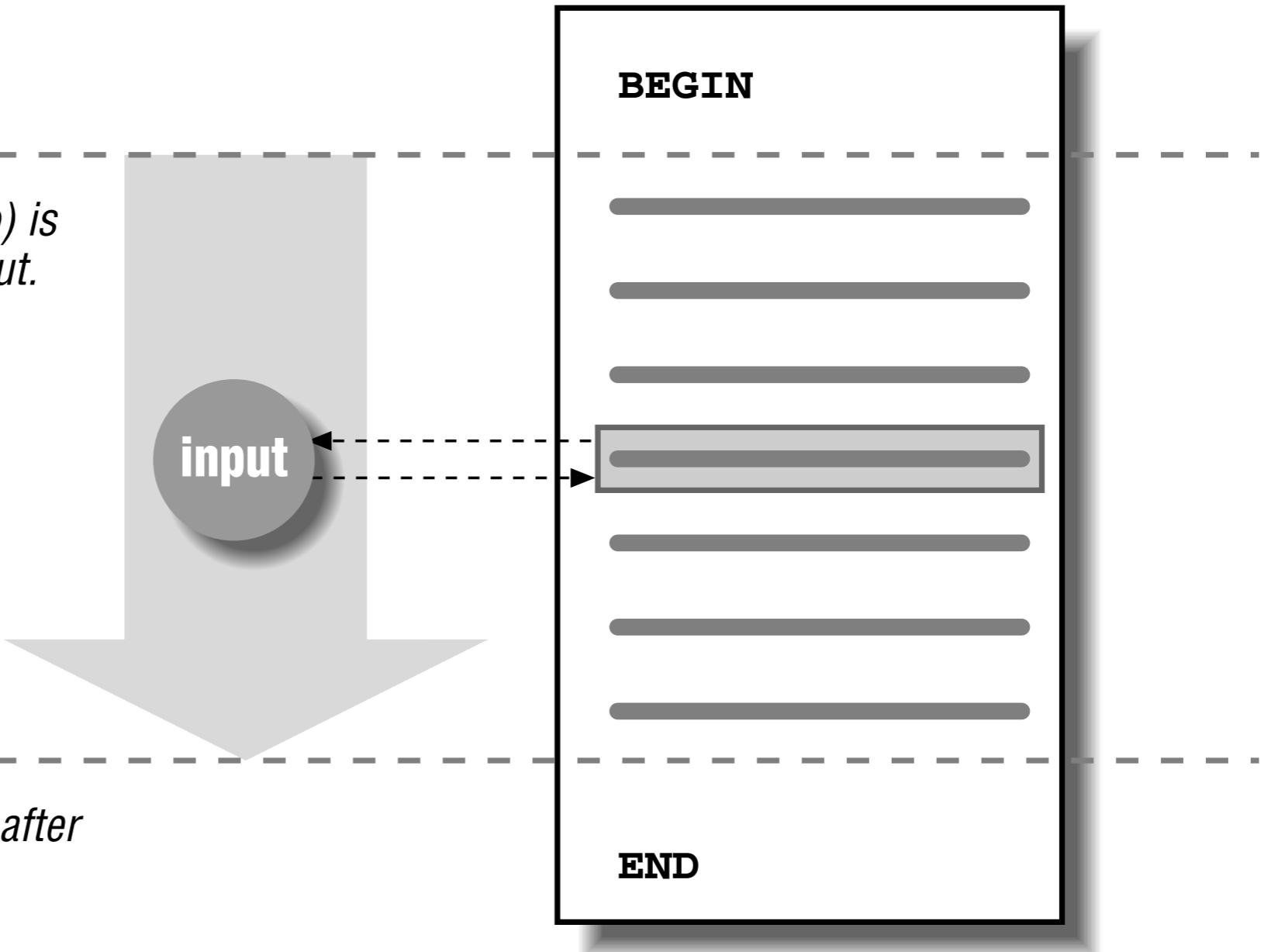
Linux

Awk

1 *1st routine is executed once before any input is read.*

2 *2nd routine (main input loop) is executed for each line of input.*

3 *3rd routine is executed once after all input is read.*



Awk & sed

Problems

- 1.Count the frequency of words in a text file.
- 2.Change a hello for bye in the 10-20 lines of a file.
- 3.Merge two files by a common field.
- 4.Create a table combining one or more fields from several files.
- 5.Count 100 most frequent 15-mer of a fasta sequence file.
- 6.Transpose of a numeric matrix.

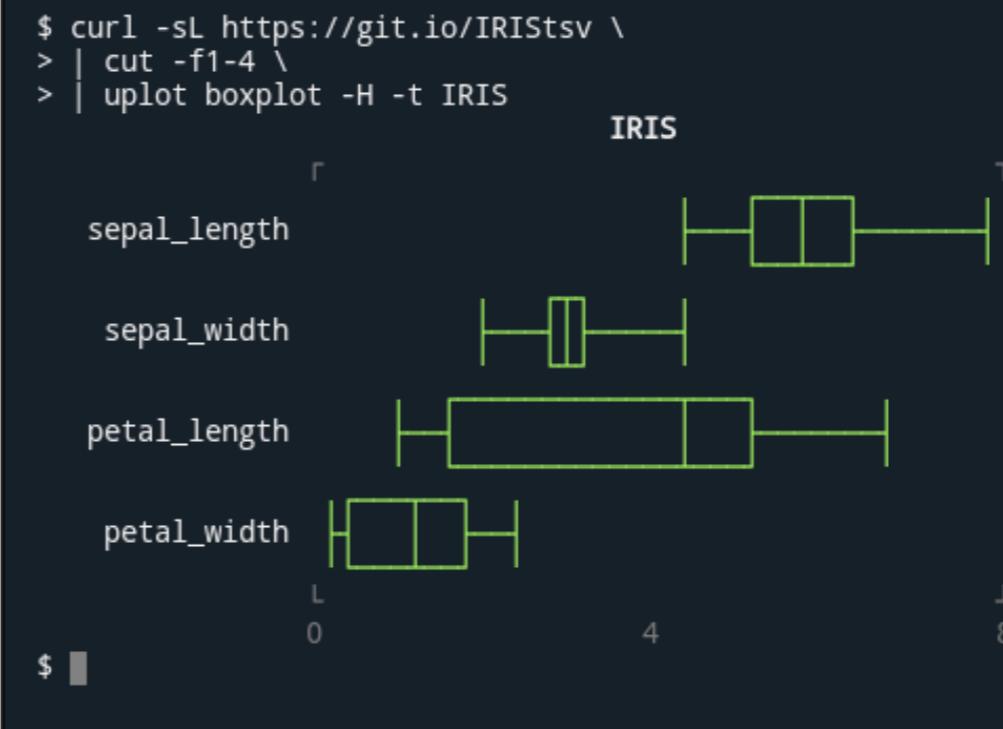
Awk and Sed Solutions

1. **awk** '{for(i=1;i<NF;i++){a[\$i]+=1;}}END{for(i in a){print i" "a[i]}}'
matrix.txt | sort -nr -k2,2
2. seq 1 100 | sed -n '20,30 p'
- 3.todo
- 4.todo
5. **sed** 's/.\\{15\\}/&\\n/g' | **awk** '{if(\$1 ~/[ACTG]/){a[\$1]++}}END{for(i in a){print i" "a[i]}}' | sort -nr -k2,2 | head -n 30
6. **awk** 'NR == 1 {n=NF; for(l=0;i<NF;i++){row[i]="\$i"} next;} {if(NF > n) {n=NF} for(i=1; i<NF; i++){row[i]=row[i]" "\$i}}END{for(i=1;i<=n; i++){print row[i]}}' matrix.txt

XSV, Youplot and csvtk

- **xsv** is a command line program for indexing, slicing, analyzing, splitting and joining CSV files.
 - <https://github.com/BurntSushi/xsv>
- **YouPlot**
 - command line tool that draws plots on the terminal.
- **Csvtk** is convenient for rapid data investigation and easily integrated into analysis pipelines.
 - <https://github.com/shenwei356/csvtk>

```
$ curl -sL https://git.io/ISLANDScsv \  
> | sort -nk2 -t, \  
> | tail -n15 \  
> | uplot bar -d, -t "Areas of the World's Major Landmasses"  
          Areas of the World's Major Landmasses  
  
          Britain 84.0  
          Honshu 89.0  
          Sumatra 183.0  
          Baffin 184.0  
          Madagascar 227.0  
          Borneo 280.0  
          New Guinea 306.0  
          Greenland 840.0  
          Australia 2968.0  
          Europe 3745.0  
          Antarctica 5500.0  
          South America 6795.0  
          North America 9390.0  
          Africa 11506.0  
          Asia 16988.0
```



Fundamentos programación: Python

Why learn to code in Python?

- Python code is designed to be readable, and hence reusable and maintainable.
 - Easy to learn, understand and remember.
 - Developer productivity (1/3 of C or C++ code).
 - Program portability (run on major operating systems)
 - A large collection of libraries.
 - Component integration (C and C++)
 - A big community.
 - Python is a general-purpose programming language that blends procedural, functional, and object-oriented paradigms.
 - Speed is a downside of Python (10-100X slower than C/C++).



- What can I do?
 - System programming.
 - GUIs
 - Internet scripting
 - Database programming
 - Rapid prototyping
 - Numeric and scientific programming
 - Machine learning.

First code in Python

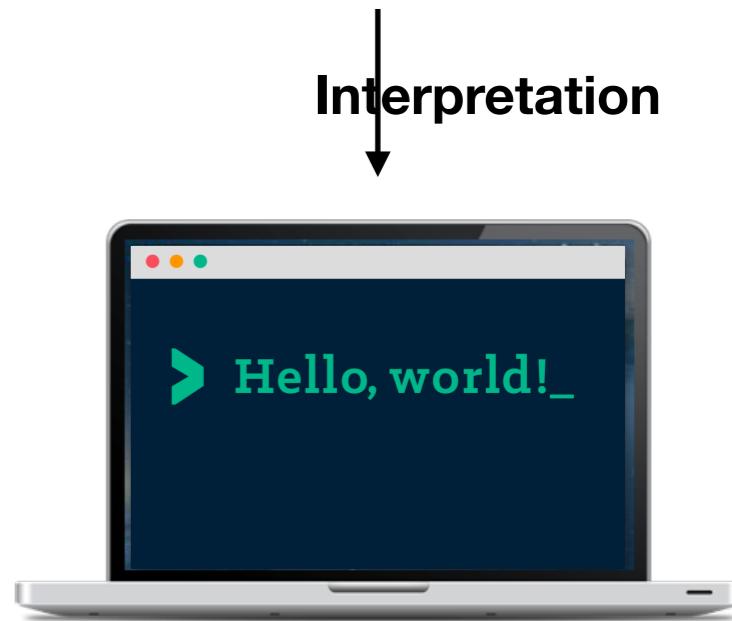
Code

Python

```
1 # file.py
2 print("Hello World")
```

Interpretation

> Hello, world!_



First code in Python

Python

```
1 #file.py
2 print("Hello, World")
```

Bash

```
1 #!/bin/bash
2 echo "Hello, World"
```

Java

```
1 //file.java
2 class HelloWorld {
3 {
4     public static void main(String args[]) {
5     }
6     System.out.println("Hello, World");
7 }
8 }
```

C

```
1 #include<stdio.h>
2 int main() {
3 {
4     printf("Hello, World");
5     return 0;
6 }
```

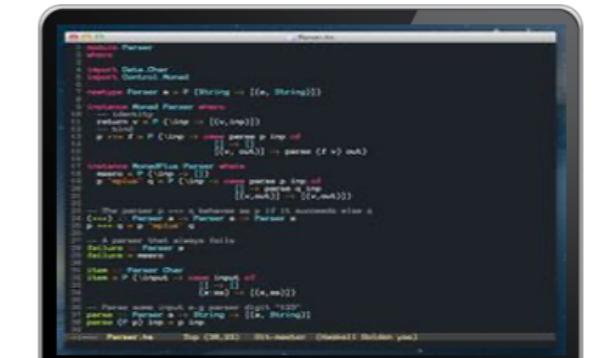
C++

```
1 //file.cpp
2 #include <iostream>
3 int main() {
4 {
5     std::cout << "Hello, World!" << std::endl;
6     return 0;
7 }
```

R

```
1 hello <- "Hello, World!"
2 print(hello)
3
```

Code



Interpretation



First code in Python



¿Qué frases entienden los computadores?

Operaciones matemáticas

- En matemáticas, un **operador** es un símbolo que se usa cuando resolvemos un problema matemático.

- + (sumar)

```
[3] 1 # suma 13  
2 6 + 7 + 4 + 5
```

22

- - (restar)

```
[ ] 1 # resta  
2 10 - 5
```

5

- x (multiplicar)

```
[ ] 1 # multiplicacion 18  
2 3 * 6
```

18

- ÷ (division)

```
▶ 1 # division  
2 8 / 3
```

2.6666666666666665

- // (division entera)

```
[ ] 1 # division entera  
2 12//3
```

4

- % (modulo)

```
▶ 1 # modulo  
2 10 % 5
```

0

- Exp (exponencial)

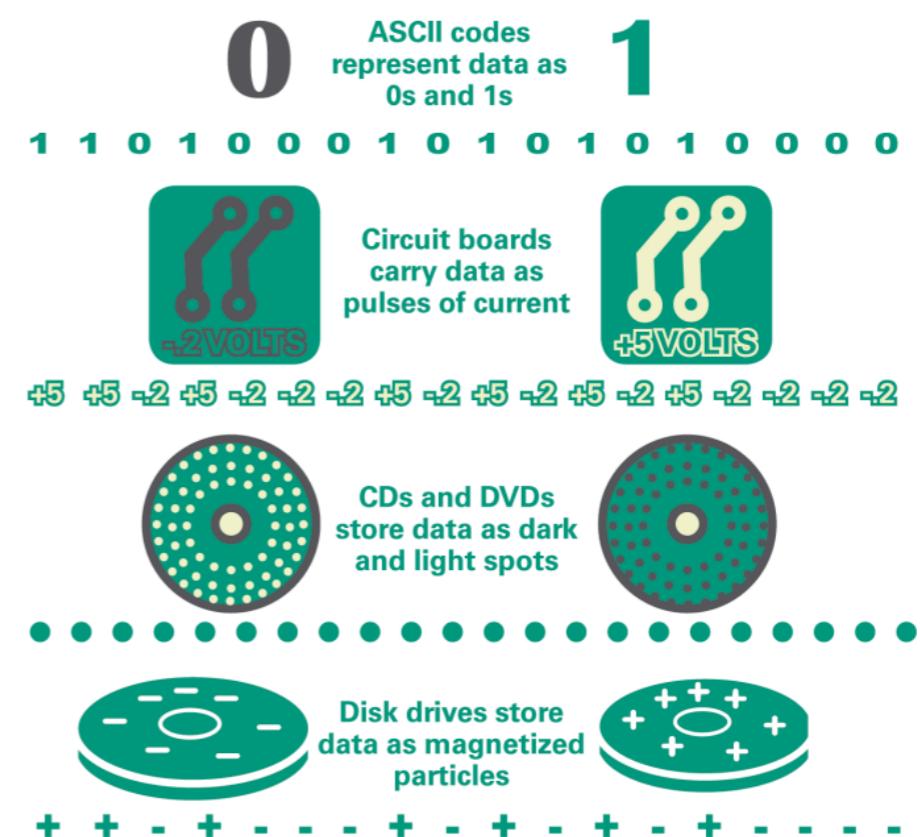
```
[4] 1 # exponencial 2 * 2 * 2  
2 5 ** 3
```

125

¿Qué frases entienden los computadores?

¿Cómo representa la información mi computador?

- Los 0 y 1 utilizados para representar datos digitales son denominados dígitos binarios — de este término obtenemos la palabra **bit** que significa **dígito binario**.
 - Un **bit** es un **0** o un **1** usado en la representación digital de datos.
 - Un archivo digital (archivo), es una colección de datos con nombre que existe en un medio de almacenamiento, como un **disco duro, CD, DVD o unidad flash**.



¿Qué frases entienden los computadores?

Representando números

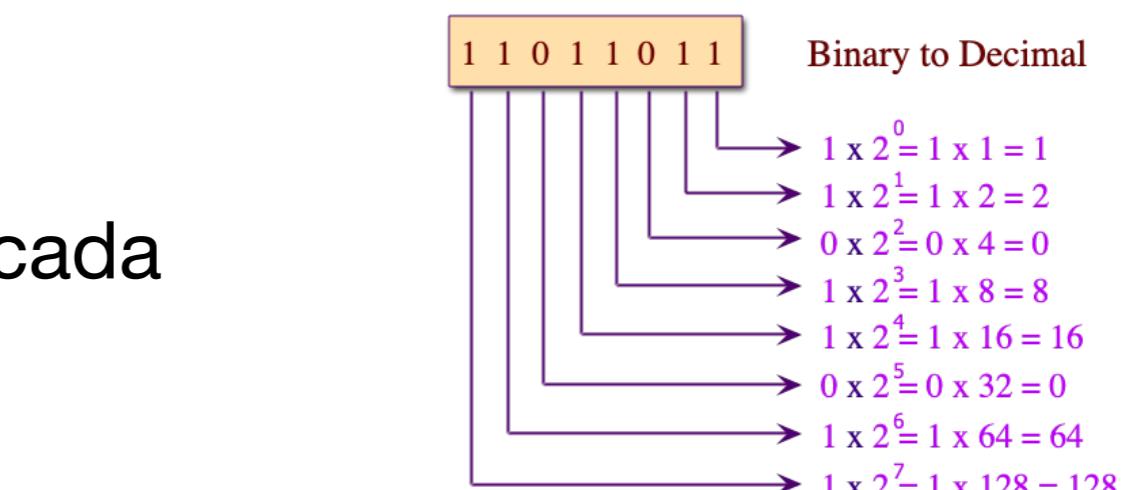
- Los **computadores** representan datos numéricos usando el sistema numérico binario, también llamado **base 2**
- El sistema numérico binario solo tiene dos dígitos: **0 y 1**.
- En python: usamos las funciones **bin** y **int** para convertir números a binario y de binarios a números.

ASCII extendido utiliza ocho bits para cada carácter (2^8 caracteres).

A es 01000001

0	->	0b0	->	0
1	->	0b1	->	1
2	->	0b10	->	2
3	->	0b11	->	3
4	->	0b100	->	4
5	->	0b101	->	5
6	->	0b110	->	6
7	->	0b111	->	7
8	->	0b1000	->	8
9	->	0b1001	->	9
10	->	0b1010	->	10
11	->	0b1011	->	11
12	->	0b1100	->	12
13	->	0b1101	->	13
14	->	0b1110	->	14
15	->	0b1111	->	15
16	->	0b10000	->	16
17	->	0b10001	->	17
18	->	0b10010	->	18
19	->	0b10011	->	19
219	->	0b11011011	->	219

```
a = bin(5)  
print(a)  
print(int(a))
```



¿Qué frases entienden los computadores?

Representando texto

ASCII extendido utiliza ocho bits para cada carácter (2^8 caracteres).

A es 01000001

Tabla ASCII

00100000	Space	00110011	3	01000110	F	01011001	Y	01101100	I
00100001	!	00110100	4	01000111	G	01011010	Z	01101101	m
00100010	"	00110101	5	01001000	H	01011011	[01101110	n
00100011	#	00110110	6	01001001	I	01011100	\	01101111	o
00100100	\$	00110111	7	01001010	J	01011101]	01110000	p
00100101	%	00111000	8	01001011	K	01011110	^	01110001	q
00100110	&	00111001	9	01001100	L	01011111	_	01110010	r
00100111	'	00111010	:	01001101	M	01100000	`	01110011	s
00101000	(00111011	:	01001110	N	01100001	a	01110100	t
00101001)	00111100	<	01001111	O	01100010	b	01110101	u
00101010	*	00111101	=	01010000	P	01100011	c	01110110	v
00101011	+	00111110	>	01010001	Q	01100100	d	01110111	w
00101100	,	00111111	?	01010010	R	01100101	e	01111000	x
00101101	-	01000000	@	01010011	S	01100110	f	01111001	y
00101110	.	01000001	A	01010100	T	01100111	g	01111010	z
00101111	/	01000010	B	01010101	U	01101000	h	01111011	{
00110000	0	01000011	C	01010110	V	01101001	i	01111100	
00110001	1	01000100	D	01010111	W	01101010	j	01111101	}
00110010	2	01000101	E	01011000	X	01101011	k	01111110	~

Python:

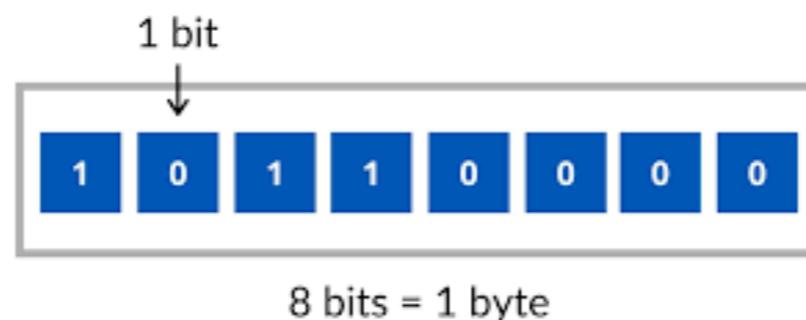
```
#texto a binario
print("A")
print(ord("A"))
print(bin(ord("A")))
print(chr(ord("A")))
```

A
65
0b1000001
A

¿Qué frases entienden los computadores?

Bits y bites

Todos los datos almacenados y transmitidos por computadores se codifican como **bits**.



UNIDAD	SÍMBOLO	EQUIVALENCIA		
Kilobyte	kB	2^{10} bytes	1024 bytes	
Megabyte	MB	2^{10} kilobytes	1024 kilobytes	
Gigabyte	GB	2^{10} megabytes	1024 megabytes	1000.000.000 bytes
Terabyte	TB	2^{10} gigabytes	1024 gigabytes	1 billón de bytes
Petabyte	PB	2^{10} terabytes	1024 terabytes	1000 billones de bytes
Exabyte	EB	2^{10} petabytes	1024 petabytes	1 trillón de bytes
Zettabyte	ZB	2^{10} exabytes	1024 exabytes	1000 trillones de bytes
Yottabyte	YB	2^{10} zettabytes	1024 zettabyte	1 cuatrillón de bytes

¿Qué frases entienden los computadores?

Explicación del concepto variable.

Las variables tiene dos partes:

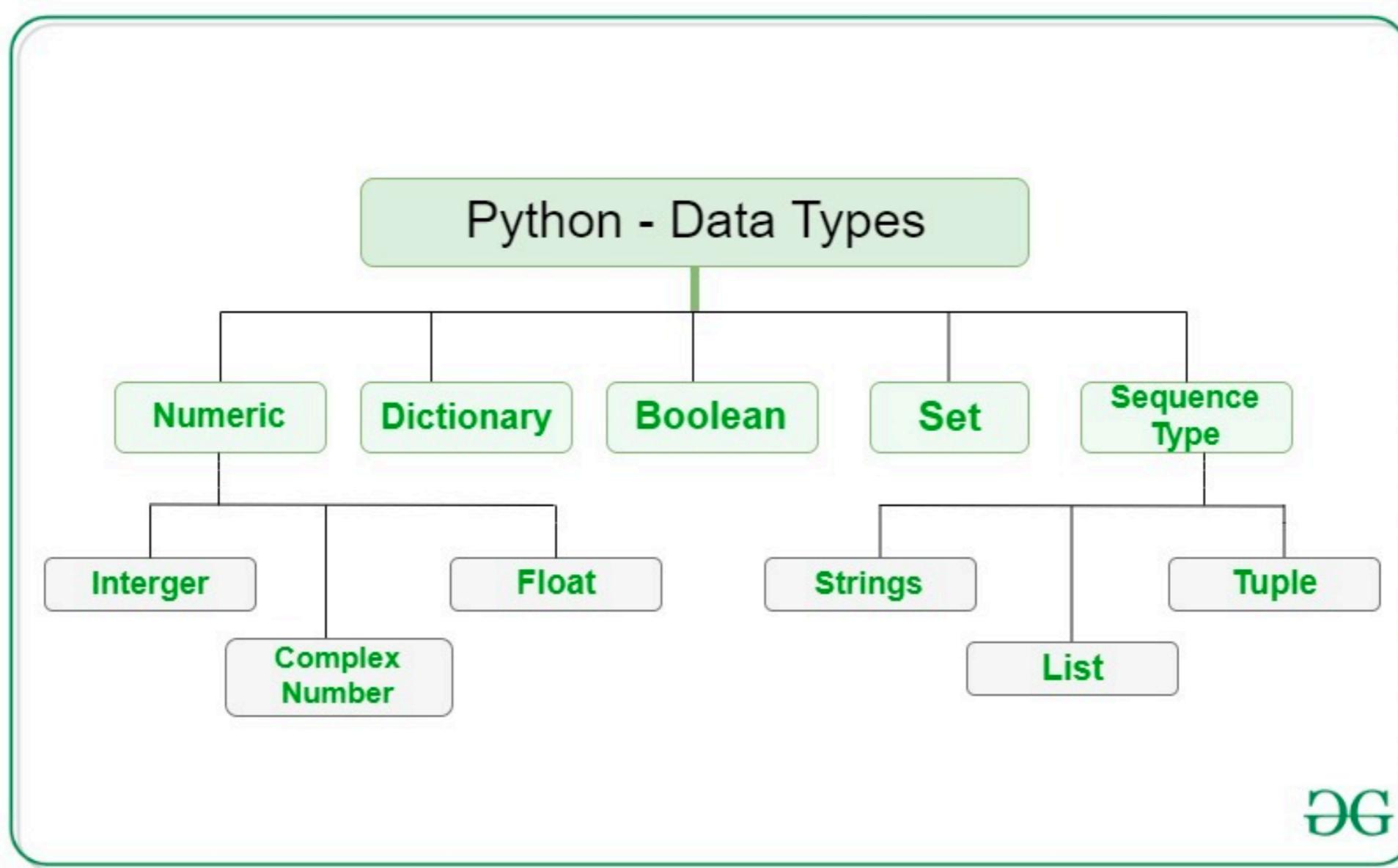
1. **El nombre.** También llamado identificador: algo que se usa para identificar (decir qué es algo). El nombre de la variable se utiliza para identificar un dato exacto.
 2. **El valor.** Valor significa cantidad o tipo. La parte de valor de una variable muestra los datos que necesitamos para realizar un seguimiento.
- Usamos un signo igual (=) para asignar (dar) un **valor** al **nombre** de una **variable**.

```
# variables
a = 5
print(a)
a = 7
print(a)
b = a
print(b)
c = "texto"
b = c
print(b)
```

¿Qué frases entienden los computadores?

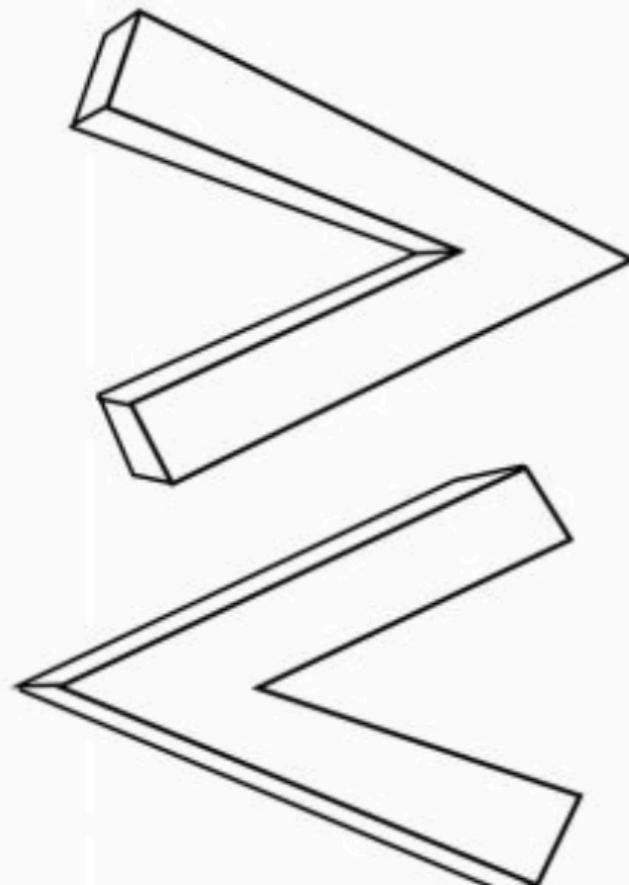
Explicación del concepto variable.

Tipos de variables en python:



Mi computador es el mejor para el juego de la verdad

Comparaciones numéricas (mayor, menor, igual) en Python.



```
1 5 > 7
```

False

```
1 5 < 7
```

True

```
1 a = 100  
2 b = 50  
3 a < b
```

False

```
1 5 == 5
```

True

```
[66] 1 5 == 6
```

False

```
[67] 1 5 is 5
```

True

```
1 5 is 6
```

False

```
1 5 > 3 and 10 < 20
```

True

```
1 a = 5  
2 a is 5 or a == 4
```

True

```
1 a >= 5
```

True

```
1 a <= 10
```

True

```
1 a = "Alex"  
2 b = "Carol"  
3 a != b
```

True

```
1 a is not b
```

True

Mi computador es el mejor para el juego de la verdad

Manejo y comparaciones de palabras en Python.

- Las palabras (strings) se utilizan para almacenar texto.
- En python se implementan como secuencias. Una secuencia es una colección/arreglo ordenado por posición de objetos idénticos (orden de izquierda-derecha).
- Strings son secuencias de un carácter (listas, tuplas).

```
[1] 1 s = 'palabra'  
2 len(s)
```

7

```
[7] 1 print(s[0]) #primer caracter de la secuencia  
2 print(s[3]) # cuarto caracter
```

p
a



```
1 # indices de secuencias comienzan en 0 .. n-1  
2 print(s[-1]) # ultimo caracter  
3 print(s[-2]) # ante-penultimo caracter
```

a
r

Strings contienen índices por posición.

Mi computador es el mejor para el juego de la verdad

Manejo y comparaciones de palabras en Python.

- Podemos extraer porciones de un string [pos1:pos2].

```
print(S[2:5])      lab
print(S[1:])       alabra
print(S[:4])       pala
print(S[:-1])     palabr
print(S[:])        palabra
```

- Concatenar strings

```
S + ' dos '
```

- Repetir strings

```
S * 4
```

- Los strings son inmutables.

```
S[0] = "P"      S = 'P' + S[1:]
print(S)
Palabra
```

Mi computador es el mejor para el juego de la verdad

Manejo y comparaciones de palabras en Python.

- Los string poseen funciones específicas.

**capitalize casfold center count encode endswith expandtabs find
format format_map index isalnum isalpha isascii isdecimal
isdigit
isidentifier islower isnumeric isprintable isspace istitle
isupper join
ljust lower lstrip maketrans partition replace rfind rindex
rjust rpartition rsplit rstrip split splitlines startswith strip
swapcase title translate upper zfill**

print(S.capitalize())	Palabra
print(S.upper())	PALABRA
print(S.find("bra"))	4
print(S.replace("bra", "BRA"))	PalaBRA
l="aaa,bbb,ccc,ddd\n"	['aaa', 'bbb', 'ccc', 'ddd\n']
print(l.split(","))	aaa,bbb,ccc,ddd
print(l.rstrip())	['aaa', 'bbb', 'ccc', 'ddd']
print(l.rstrip().split(","))	

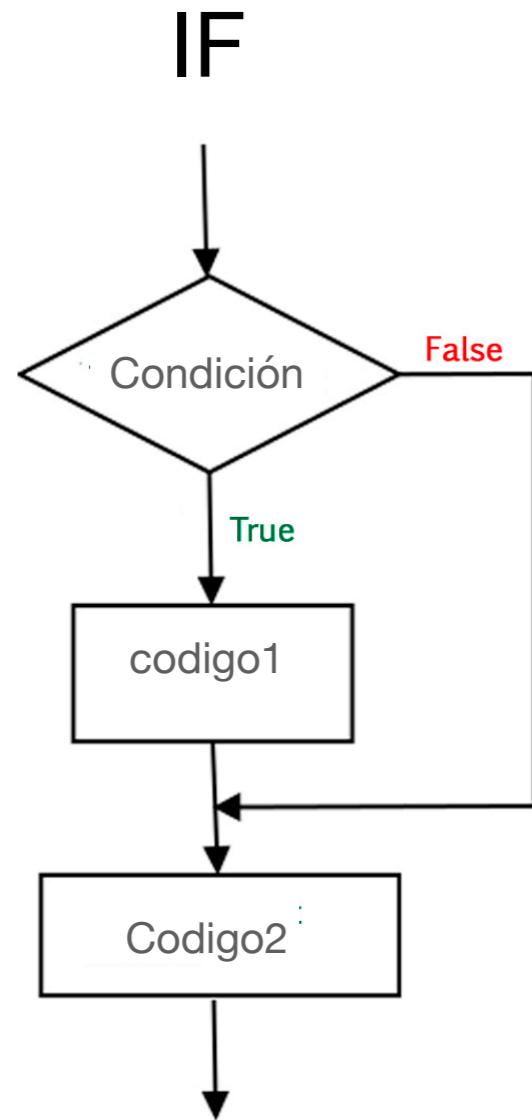
A mi computador le gusta tomar decisiones

Presentación de instrucciones de decisión (if, elif, else) en Python



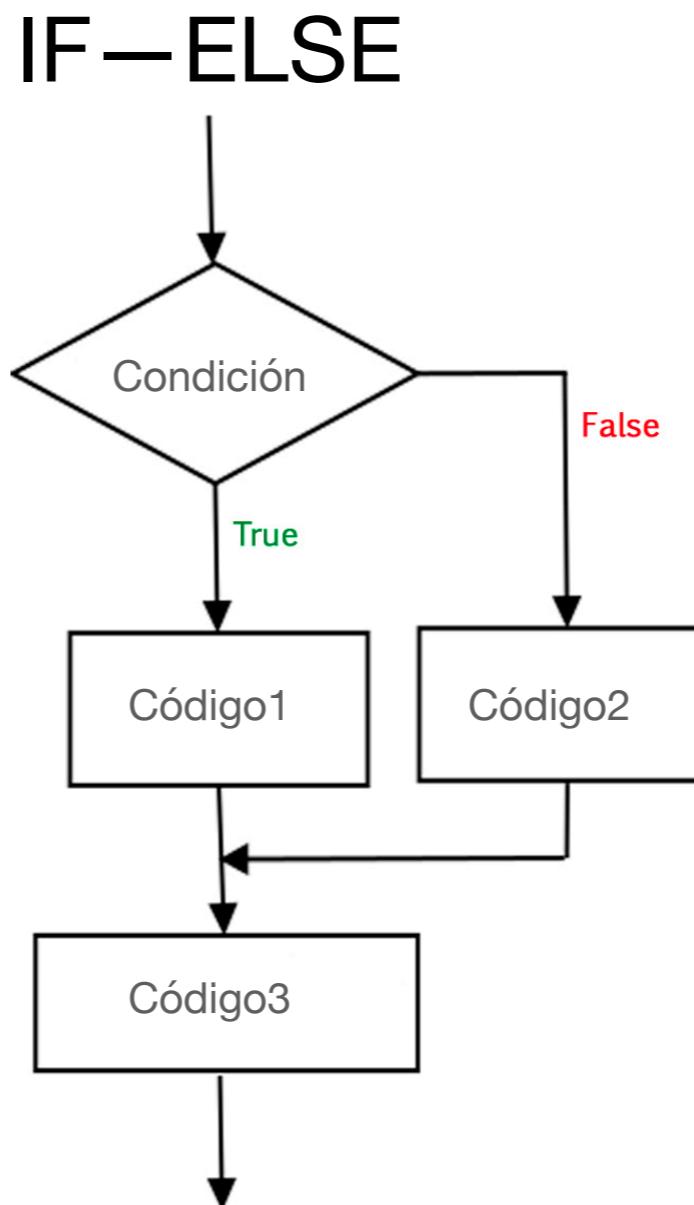
A mi computador le gusta tomar decisiones

Presentación de instrucciones de decisión (if, elif, else) en Python



```
a = 11
if(a > 10):
    print("el valor es mayor a 10")

print("el valor es:",a)
```

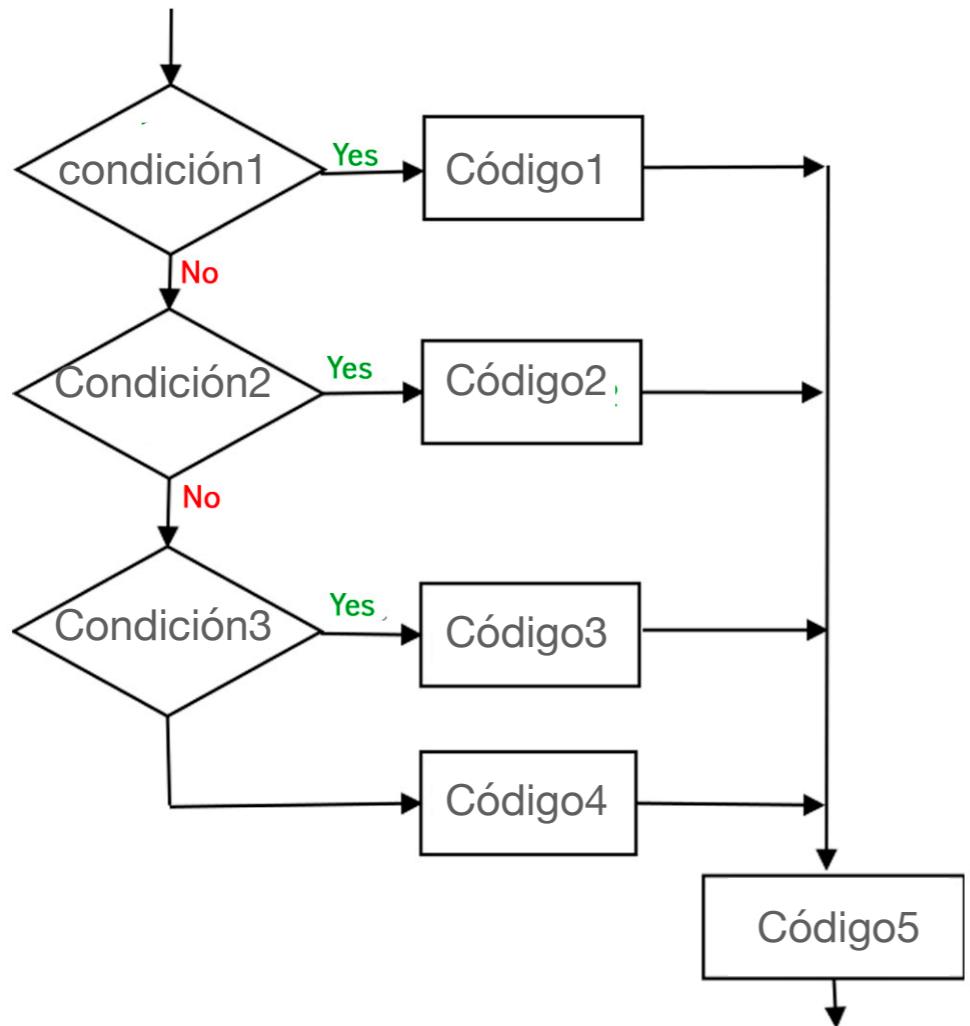


```
texto = "Alex"
if(texto == "Alex"):
    print("Hola Alex")
else:
    print("No eres alex")
```

A mi computador le gusta tomar decisiones

Presentación de instrucciones de decisión (if, elif, else) en Python

IF-elif-else



```
#if-elif-else
nombre= "Alex"
if(nombre == "Alex"):
    print("Hola Alex")
    print("Se que te gusta jugar al futbol")
elif(nombre == "Roberto"):
    print("Hola Roberto")
    print("Me contaron que te gusta leer libros")
elif(nombre == "Camila"):
    print("Hola Camila")
    print("Se que te gusta programar")
else:
    print("Hola ",nombre)
    print("No se nada de ti")

print("Bienvenido")
```

A mi computador le gusta tomar decisiones

Presentación de comparaciones con múltiples decisiones.

```
edad = 20
nombre = "Carlos"
ciudad = "Talca"

if(nombre == "Carlos"):
    if(ciudad == "Rancagua"):
        if(edad < 20):
            print("Hola Carlos de Rancagua, tienes menos de 20 años")
        else:
            print("Hola Carlos de Rancagua, tienes más de 20 años")
    else:
        print("La persona no es de rancagua")
else:
    print("El nombre de la persona no es Carlos")
```

Que hace este código?

```
edad = 20
nombre = "Alex"
ciudad = "Rancagua"

if(nombre == "Carlos" and ciudad == "Rancagua"):
    if(edad < 20):
        print("Hola Carlos de Rancagua, tienes menos de 20 años")
    else:
        print("Hola Carlos de Rancagua, tienes más de 20 años")
else:
    if(nombre == "Carlos"): print("es Carlos pero no es de Rancagua")
    if(ciudad == "Rancagua"): print("No es Carlos, pero es de Rancagua")
```

A mi computador le gusta tomar decisiones

Controlando el tiempo en el computador.

`sleep(...)`

`sleep(segundos)`

Retrasar la ejecución durante un número determinado de segundos. El argumento puede ser un número de punto flotante para una precisión de subsegundos.

```
import time

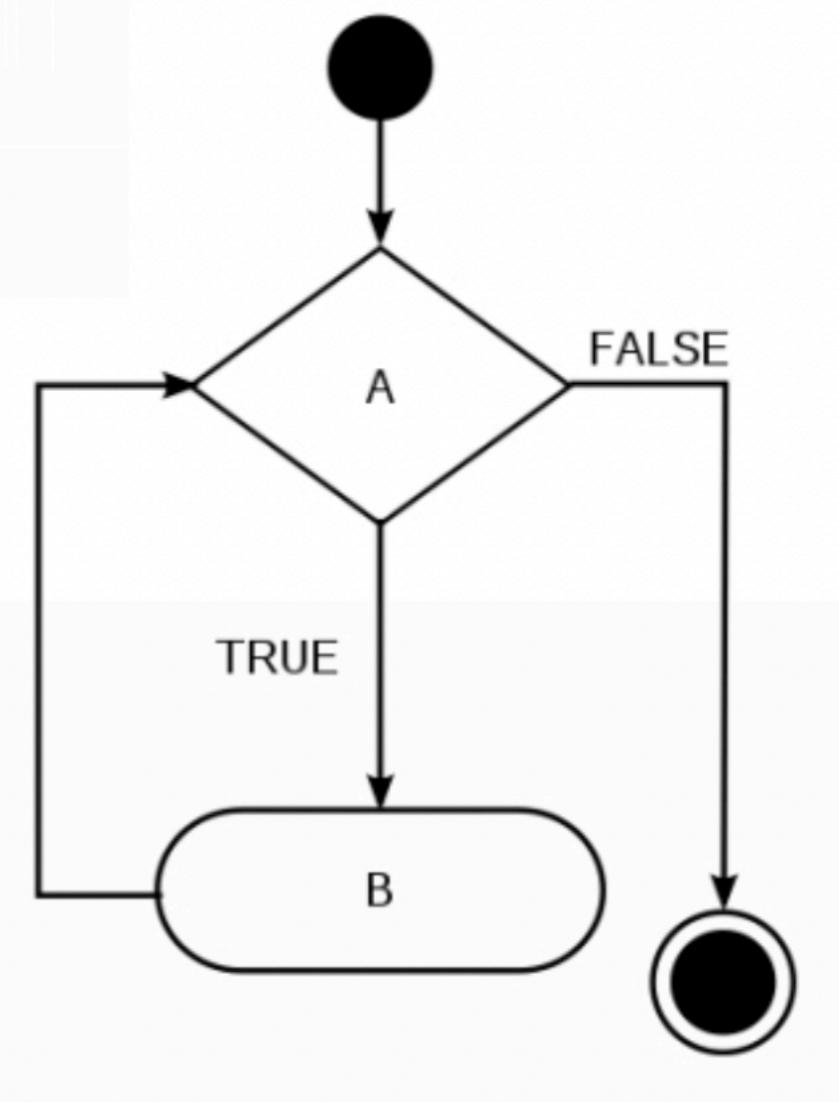
a = 10
nombre = input("Ingresa un nombre:")
if(nombre == "Roberto"):
    time.sleep(10)
    print("Roberto esta esperando")
else:
    time.sleep(1)
    print("Hola", nombre)
```

Que hace este código?

Mi computador es el mejor para repetir instrucciones.

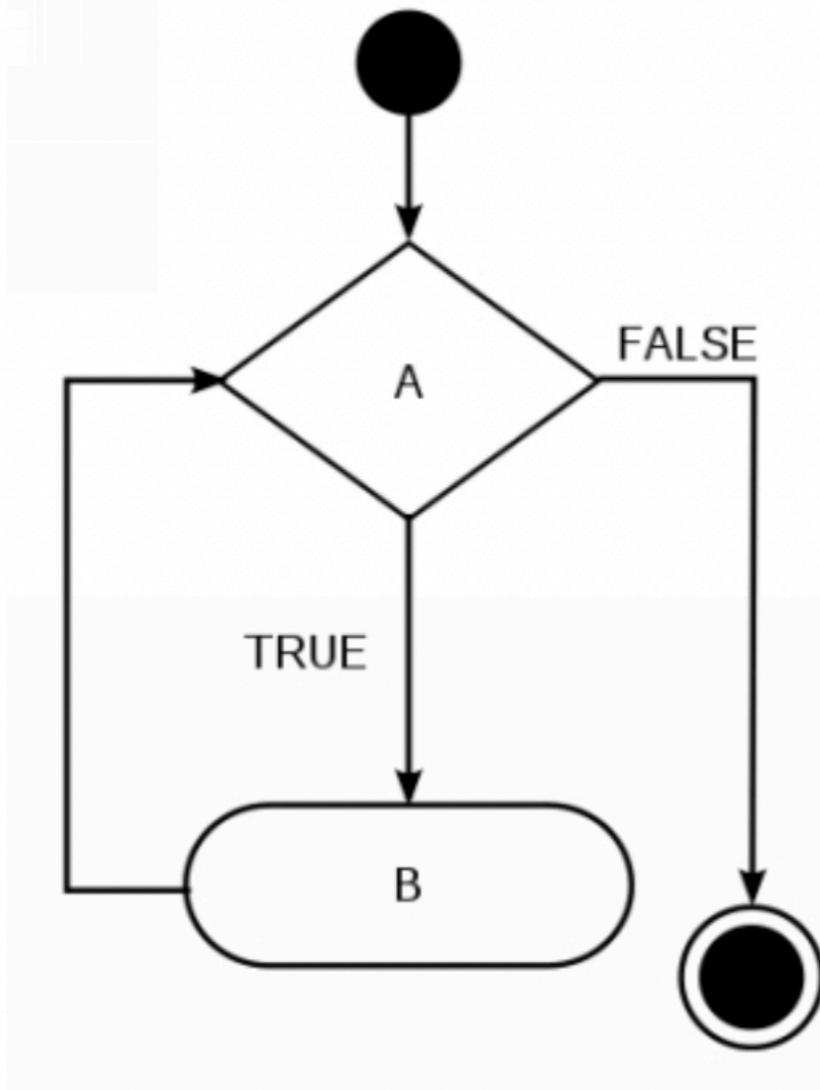
Presentación de instrucción de iteración “While” en Python

- Una computadora puede seguir haciendo lo mismo una y otra vez para siempre.
- Un bucle (loop) es un comando para repetir algo hasta que suceda algo más. Da vueltas y vueltas, una y otra vez, como un círculo.
- Uno de los tipos de bucles más populares es el bucle **while**.
 - Mientras _____ es Verdadero, haz _____



Mi computador es el mejor para repetir instrucciones.

Presentación de instrucción de iteración “While” en Python

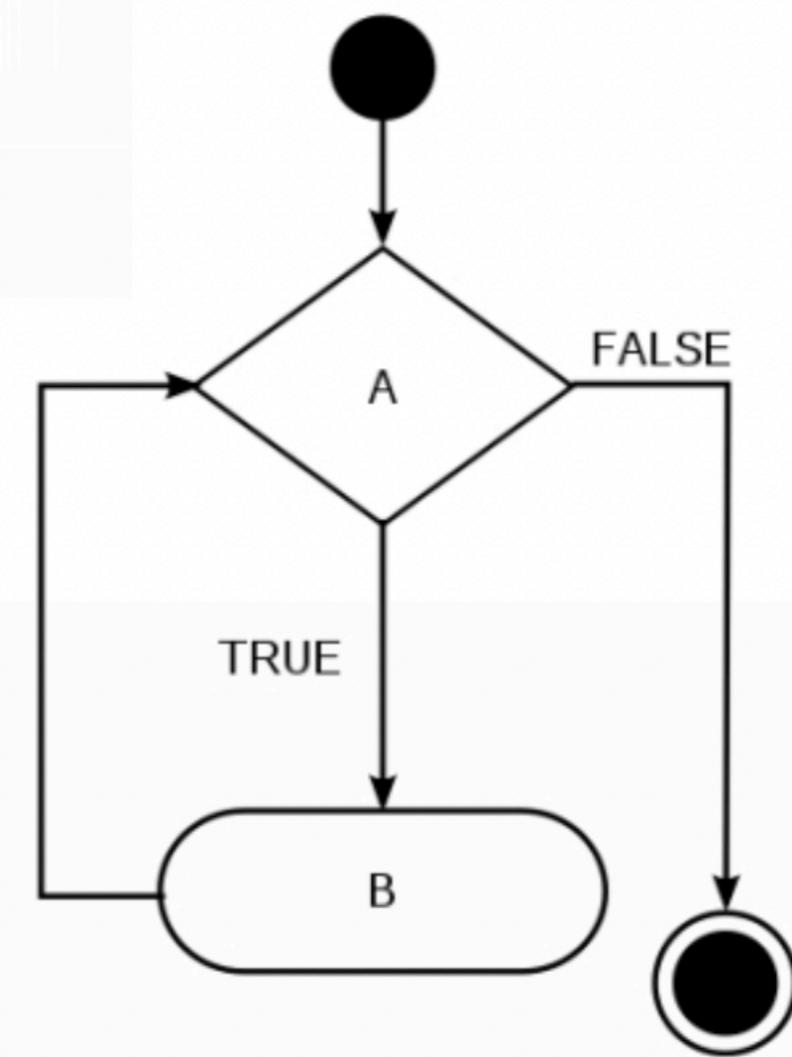


```
while 2 > 1:  
    print("Hola --- no puedo parar...")  
  
i = 1  
while i < 5:  
    print(i)  
  
i = 1  
while i < 5:  
    print(i)  
    i=i+1  
  
i = 1  
while i > 5:  
    print(i)  
    i=i+1
```

Mi computador es el mejor para repetir instrucciones.

Presentación de instrucción de iteración “For” en Python

- Con el ciclo **while**, no especificamos (decimos claramente) cuántas veces repetir el ciclo. No dijimos, repite 5 veces..
- Con el ciclo **for**, especificamos cuántas veces repetir el ciclo.
- Una de las funciones que podemos usar con bucles for es **range()**. La función **range()** se puede usar para crear una lista de números dentro de un cierto rango.



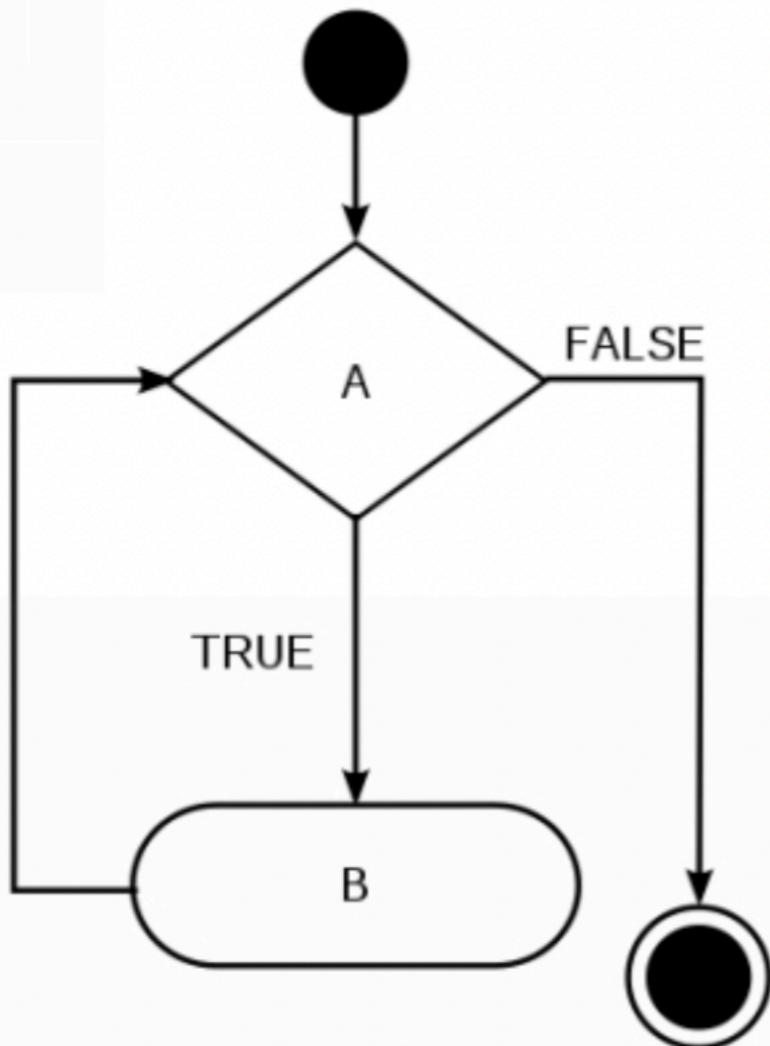
range(3)



[0,1,2]

Mi computador es el mejor para repetir instrucciones.

Presentación de instrucción de iteración “For” en Python



```
for i in range(1,10):  
    print(i)
```

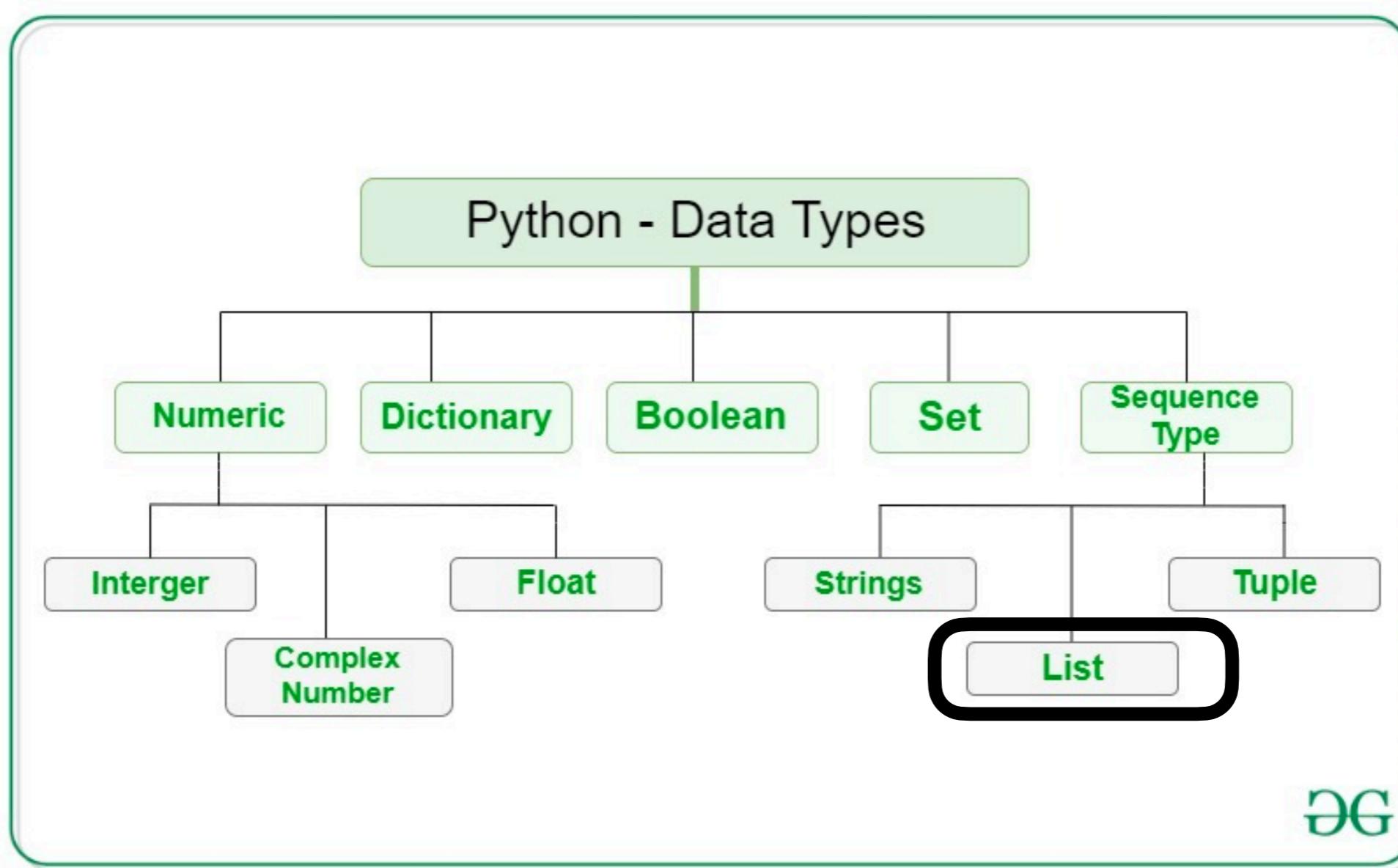
```
S="palabra"  
for i in S:  
    print(i)
```

```
for i in S:  
    for j in range(10):  
        print(i,j)
```

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

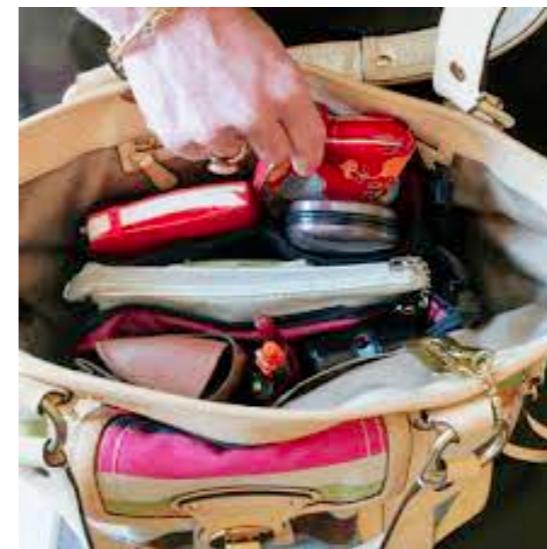
Tipos de variables en python:



Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

- Una lista, se define como una colección de elementos en un orden determinado.
- Ejemplos: letras del alfabeto, dígitos del 0 al 9, o los nombres de todas las personas de tu familia.
- Sin embargo, los elementos de una lista no tienen que estar necesariamente relacionados de ninguna manera en particular.



Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Creando una lista



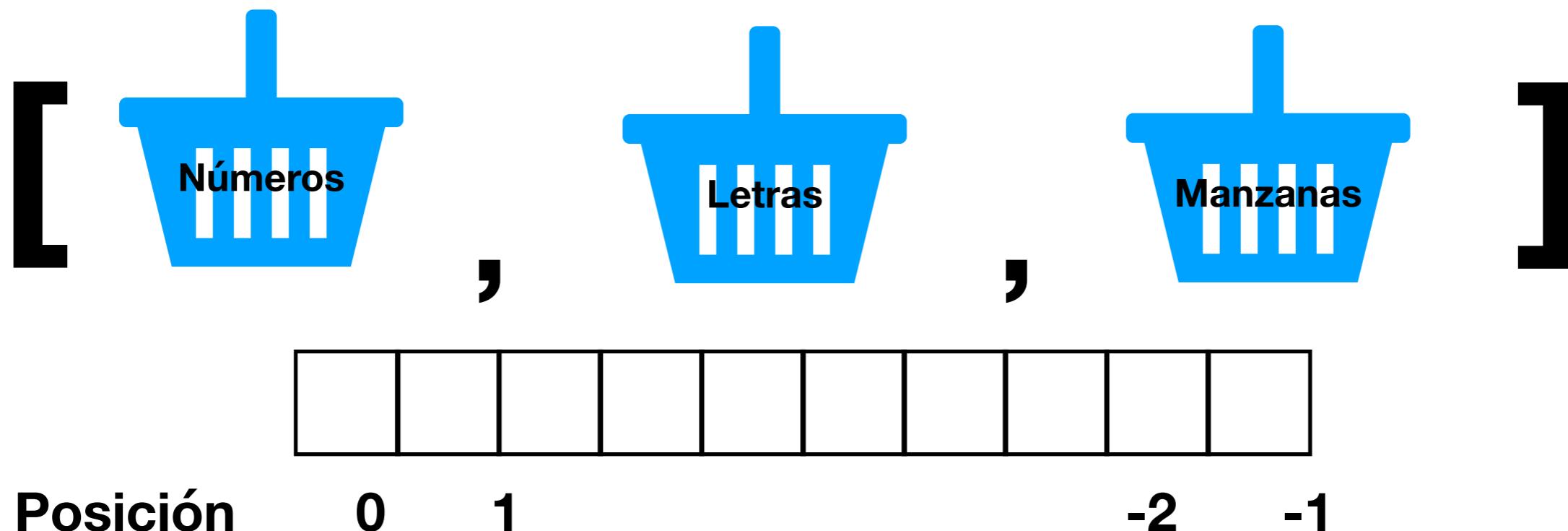
```
frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
print(frutas)
```

```
['manzanas', 'peras', 'platanos', 'frutillas']
```

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Como accedemos a los elementos en una lista?



```
# acceso a los elementos en una lista
frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
mensaje = "Mi fruta favorita son las/los " + frutas[0].title() + "."
print(mensaje)
```

↳ Mi fruta favorita son las/los Manzanas.

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Como modificamos los elementos en una lista?



```
▶ # modificar elementos en una lista  
frutas = ['manzanas', 'peras', 'platanos', 'frutillas']  
print(frutas)  
frutas[3] = "naranjas"  
print(frutas)
```

```
⇨ ['manzanas', 'peras', 'platanos', 'frutillas']  
['manzanas', 'peras', 'platanos', 'naranjas']
```

Las listas son mutables

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Como agregamos los elementos en una lista?



```
# añadir elementos en una lista
frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
print(frutas)
frutas.append('naranjas')
print(frutas)

['manzanas', 'peras', 'platanos', 'frutillas']
['manzanas', 'peras', 'platanos', 'frutillas', 'naranjas']
```

```
# añadir elementos en una posición específica a una lista
frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
print(frutas)
frutas.insert(2, 'naranjas')
print(frutas)

['manzanas', 'peras', 'platanos', 'frutillas']
['manzanas', 'peras', 'naranjas', 'platanos', 'frutillas']
```

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Como eliminamos los elementos en una lista?



```
# eliminar elementos en una posición específica a una lista  
frutas = ['manzanas', 'naranjas', 'peras', 'platanos', 'frutillas']  
print(frutas)  
del frutas[1]  
print(frutas)
```

```
[ 'manzanas', 'naranjas', 'per  
[ 'manzanas', 'peras', 'platan
```

```
# eliminar elementos con función pop  
frutas = ['manzanas', 'naranjas', 'peras', 'platanos', 'frutillas']  
print(frutas)  
fruta_eliminada= frutas.pop()  
print(frutas)  
frutas.remove('peras')  
print(frutas)
```

```
[ 'manzanas', 'naranjas', 'peras', 'platanos', 'frutillas']  
[ 'manzanas', 'naranjas', 'peras', 'platanos']  
[ 'manzanas', 'naranjas', 'platanos']
```

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Como ordenamos los elementos en una lista?



- Generalmente, las listas se crean de manera **dinámica**, por lo que no siempre puedes controlar el orden en el que tus usuarios proporcionan sus datos.

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Como ordenamos los elementos en una lista?



```
▶ frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
print(frutas)
frutas.sort()
print(frutas)

['manzanas', 'peras', 'platanos', 'frutillas']
['frutillas', 'manzanas', 'peras', 'platanos']
```

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Recorriendo los elementos de una lista?



```
▶ 1 frutas = ['manzanas', 'sandía', 'peras', 'platanos', 'frutillas']
  2 print(frutas)
  3 for i in frutas:
  4     print(i)
```

```
[ 'manzanas', 'sandía', 'peras', 'platanos', 'frutillas' ]
manzanas
sandía
peras
platanos
frutillas
```

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Errores comunes con listas.



- Un error de índice (IndexError) significa que Python no puede averiguar el índice solicitado.
- Un error de tipo ValueError puede producirse al tratar de remover un valor que no se encuentra en la lista.

Mi computador es el mejor para repetir instrucciones.

¿Cómo manejamos más de una variable?, listas en Python.

Errores comunes con listas.

```
[20] #errores comunes de indice
     frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
     print(frutas)
     print(frutas[-5])

['manzanas', 'peras', 'platanos', 'frutillas']

-----
IndexError                                     Traceback (most recent call last)
<ipython-input-20-b86b898bb2dc> in <module>()
      1 frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
      2 print(frutas)
----> 3 print(frutas[-5])

IndexError: list index out of range
```

SEARCH STACK OVERFLOW

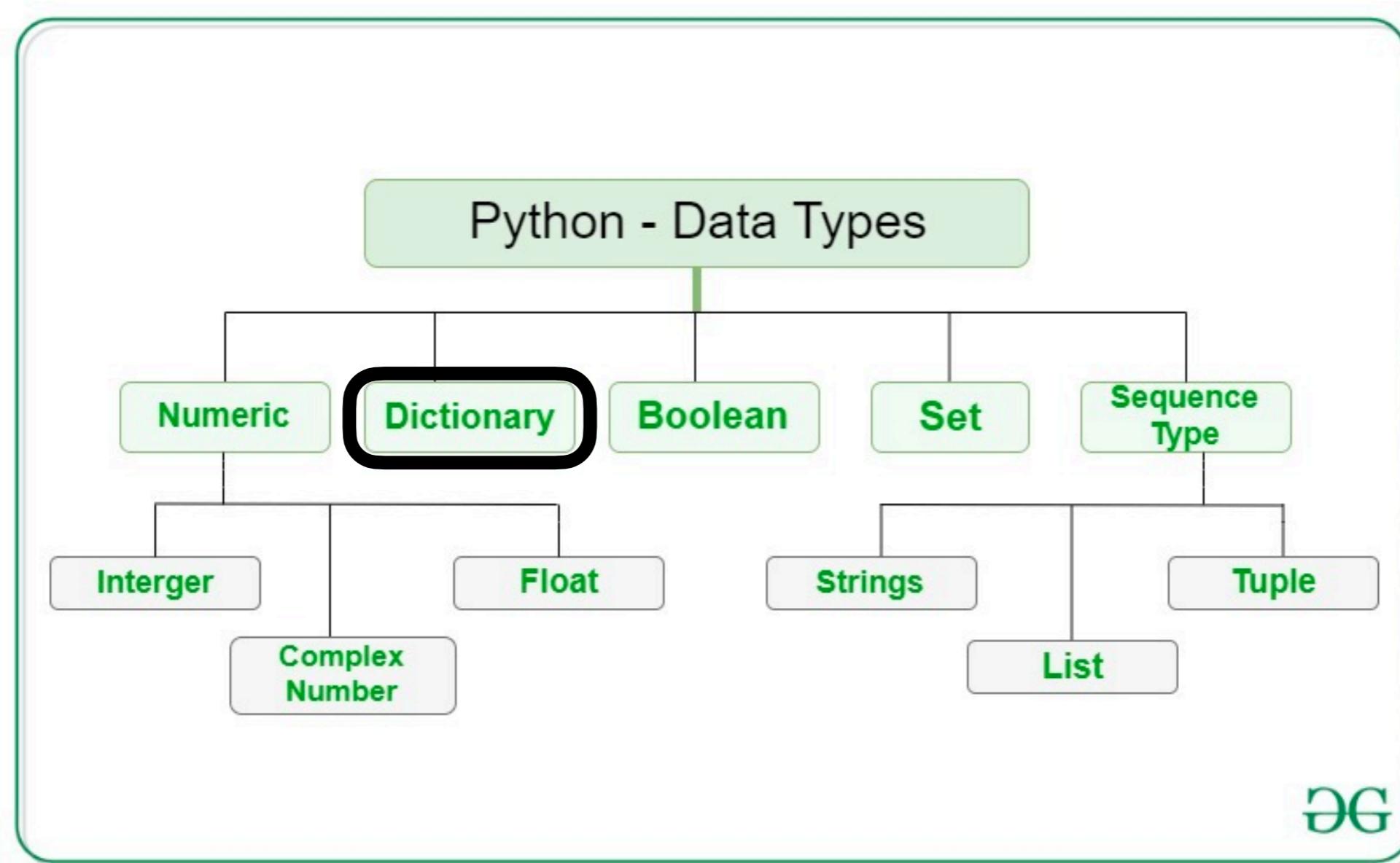
```
▶ #errores comunes de valor
    frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
    print(frutas)
    del frutas[5]
    print(frutas)

[] ['manzanas', 'peras', 'platanos', 'frutillas']

-----
IndexError                                     Traceback (most recent call last)
<ipython-input-21-f0feb5a39f37> in <module>()
      2 frutas = ['manzanas', 'peras', 'platanos', 'frutillas']
```

Mi computador sabe cómo usar un diccionario.

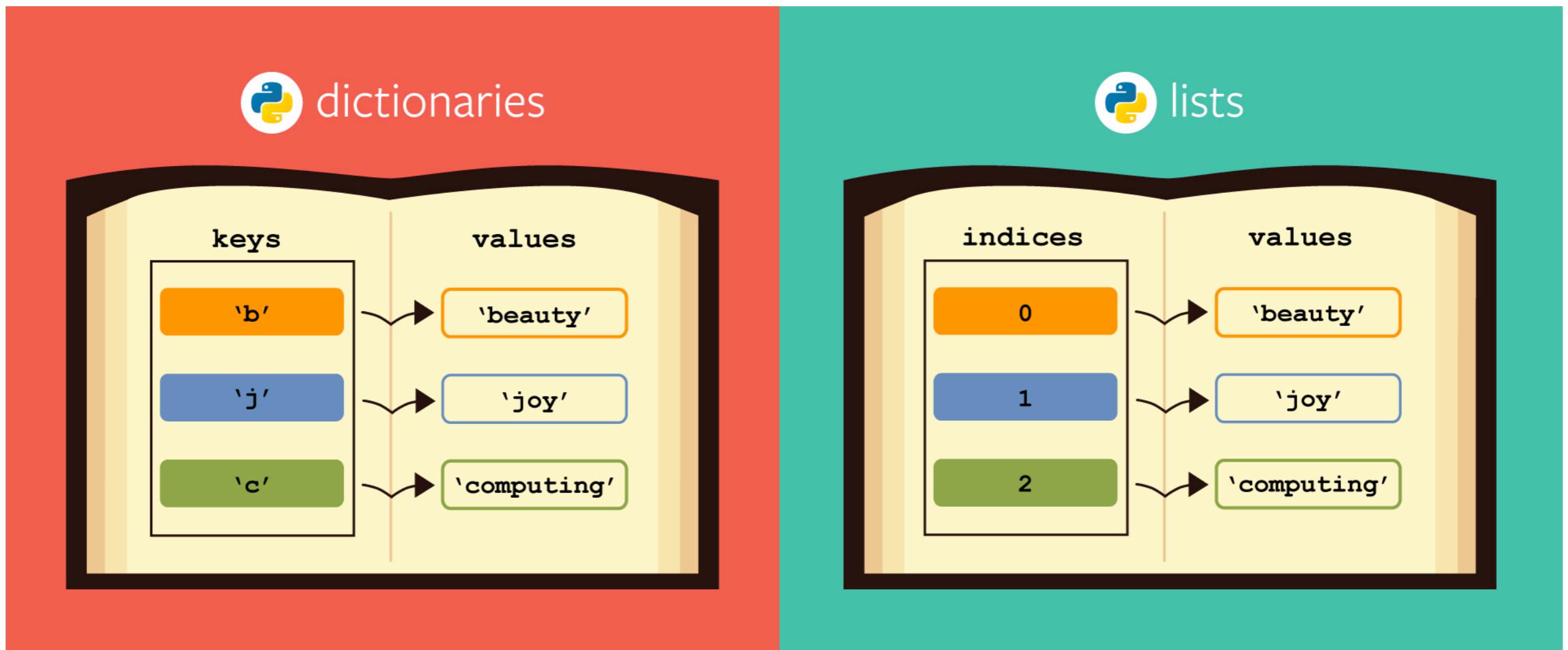
Presentación de diccionarios “hash” en python.



Mi computador sabe cómo usar un diccionario.

Presentación de diccionarios “hash” en python.

Qué es un diccionario?



Mi computador sabe cómo usar un diccionario.

Presentación de diccionarios “hash” en python.

- En Python, un diccionario es un tipo especial de lista.
- Clave: el nombre
- Valor: La cantidad o tipo de algo.

Keys
Values

```
dictionary = { "a" : 1, "b" : 2 }
```



Mi computador sabe cómo usar un diccionario.

Presentación de diccionarios “hash” en python.

▼ Diccionarios vs listas

```
[27] 1 L=list()
2 n=10000000
3 # creamos una lista con 10M numeros
4 for i in range(n):
5 | L.append(i)
6 # creamos un diccionario con 10M numeros
7 D=dict()
8 for i in range(n):
9 | D[i]=str(i)
```

```
✓ [43] 1 import time
2 start = time.time()
3 f=0
4 for i in range(100):
5 | if random.randint(0,n) in L:
6 |   f=f+1
7 end = time.time()
8 print("Lista",end - start)
```

Lista 7.328480243682861

Diccionario es ~14000 veces más rápido que la lista

```
✓ 0s
1 import time
2 start = time.time()
3 f=0
4 for i in range(100):
5 | if random.randint(0,n) in D.keys():
6 |   f=f+1
7 end = time.time()
8 print("Diccionario",end - start)
```

Lista



Guardar

Buscar



Mi computador sabe cómo usar un diccionario.

Operaciones con diccionarios (agregar, buscar, borrar) en Python.

- clear
- copy
- fromkeys
- get
- items
- keys
- pop
- popitem
- setdefault
- update
- values



```
1 #crear diccionario
2 D=dict()
3 NP={'carlos':1234,'roberto':6655,'camila':1414,'andrea':1212}
4 #acceder por clave
5 print(NP['andrea'])
6 #borra el diccionario
7 NP.clear()
8 NP={'carlos':1234,'roberto':6655,'camila':1414,'andrea':1212}
9 #copiar un diccionario
10 D=NP.copy()
11 print(D)
12 #obtener el valor por clave
13 print(NP.get("roberto"))
14 #items
15 print(NP.items())
16 #claves
17 print(NP.keys())
18 #valores
19 print(NP.values())
20 #eliminar valores por clave
21 a = NP.pop("camila")
22 print(NP, a)
```

↳ 1212
{'carlos': 1234, 'roberto': 6655, 'camila': 1414, 'andrea': 1212}
6655
dict_items([('carlos', 1234), ('roberto', 6655), ('camila', 1414), ('andrea', 1212)])
dict_keys(['carlos', 'roberto', 'camila', 'andrea'])
dict_values([1234, 6655, 1414, 1212])
{'carlos': 1234, 'roberto': 6655, 'andrea': 1212} 1414

Mi computador sabe cómo usar un diccionario.

Operaciones con diccionarios (agregar, buscar, borrar) en Python.

```
# crear bd usuarios
NP={'carlos':1234,'roberto':6655,'camila':1414,'andrea':1212}
# obtener usuario y password
nombre=input("ingrese nombre:")
password=int(input("ingrese pass"))
if nombre in NP.keys():
    if NP[nombre] == password:
        print("Welcome",nombre)
    else:
        print("Password incorrecto")
else:
    print("Usuario no registrado")
```

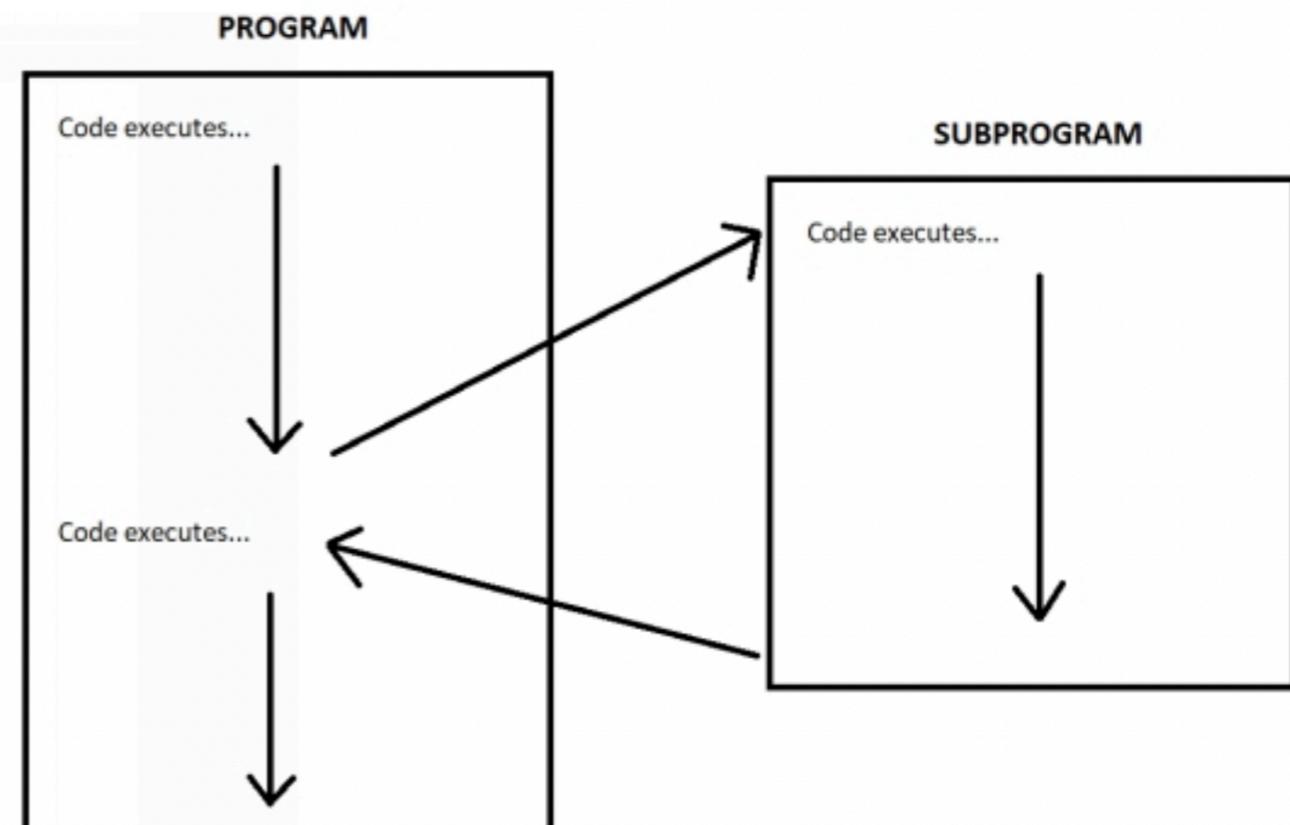
Que hace este código?

¿Puedo agregar funciones a mi computador?

No es necesario programar la rueda una y otra vez (concepto función)

- Una función es el trabajo de alguien.
 - Ejemplo : Bomberos combatir incendios.
- En programación una función es un bloque de código que hace una cosa particular.
- Lo interesante de las funciones es que se pueden reutilizar.
- Para hacer una función en Python usamos el comando **def**.
 - Nombre parametros return
- Funciones también se llaman subprogramas

```
def restar(a,b):  
    return a - b  
#llamando la nueva funcion  
print(restar(5,2))  
print(restar(9,5))
```



¿Puedo agregar funciones a mi computador?

No es necesario programar la rueda una y otra vez (concepto función)

```
1 def xxx(n):
2     i=0
3     t=0
4     while i < n:
5         t=t+i
6         i=i+1
7
8     return t
9
10 print(xxx(10))
11 print(xxx(6))
```

45
15

```
1 def zzzz(lista):
2     total=0
3     for i in lista:
4         total=total+i
5     return total/len(lista)
6
7 L=[1,2,5,6,9,10]
8
9 print(zzzz(L))
```

5.5

```
1 def yyyy(lista):
2     m = -1
3     for i in lista:
4         if(i > m):
5             m=i
6     return m
7
8 print(yyyy(L))
```

10

Que hacen las funciones?

Una serpiente que programa?

GitHub

The screenshot shows a GitHub repository page for 'adigenova/tpp'. The repository is public and contains 1 branch and 0 tags. The most recent commit was made by 'adigenova' adding class 1, 4 hours ago. The repository description is 'Curso Introductorio de Python para alumnos de Tercero y cuarto Medio'. The README.md file contains the following content:

```
Todas y Todos Podemos Programar (TTP)

Curso Introductorio de Python para alumnos de Tercero y cuarto Medio.

Programa del curso

Programa que detalla el contenido de cinco clases teóricas y cinco talleres prácticos.

Bitacora de clases

Archivo guía con el contenido y código publicados en el repositorio GitHub del taller.
```

The repository has 0 stars, 1 watching, and 0 forks. It also lists releases, packages, and languages used.

<https://github.com/adigenova/tpp>