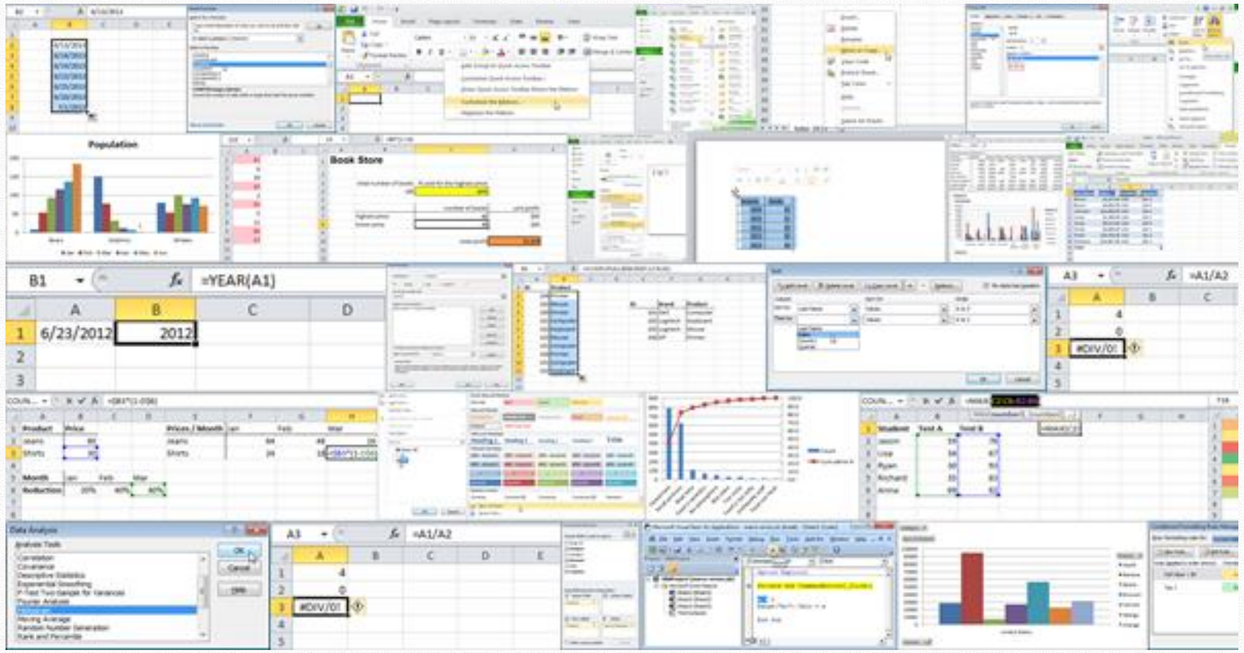
https://www.youtube.com/watch?v=hgI0p1zf31k

pwc

Universidad de Navarra

Un Excel complejo no es un algoritmo



Sino que una oportunidad para buenos consultores y organización a través de algoritmos!

Escuchar el siguiente Podcast: https://talkpython.fm/episodes/show/200/escaping-excel-hell-with-python-and-pandas

Why algorithms are called algorithms

BBC Ideas = https://www.youtube.com/watch?v=oRkNaF0QvnI

EJERCICIO: Jugar el juego de las RANAS:
http://mfalonso.pythonanywhere.com/ranas

Escribir los pasos para resolver el problema – los movimientos

Conclusión:

- Un algoritmo es un conjunto de instrucciones o reglas definidas y no-ambiguas, ordenadas y finitas que permite, típicamente, solucionar un problema, realizar un cómputo, procesar datos y llevar a cabo otras tareas o actividades. Un algoritmo tiene un estado inicial y una entrada, sigue pasos sucesivos y llega a un estado final obtiendo una solución.

Ordenadores

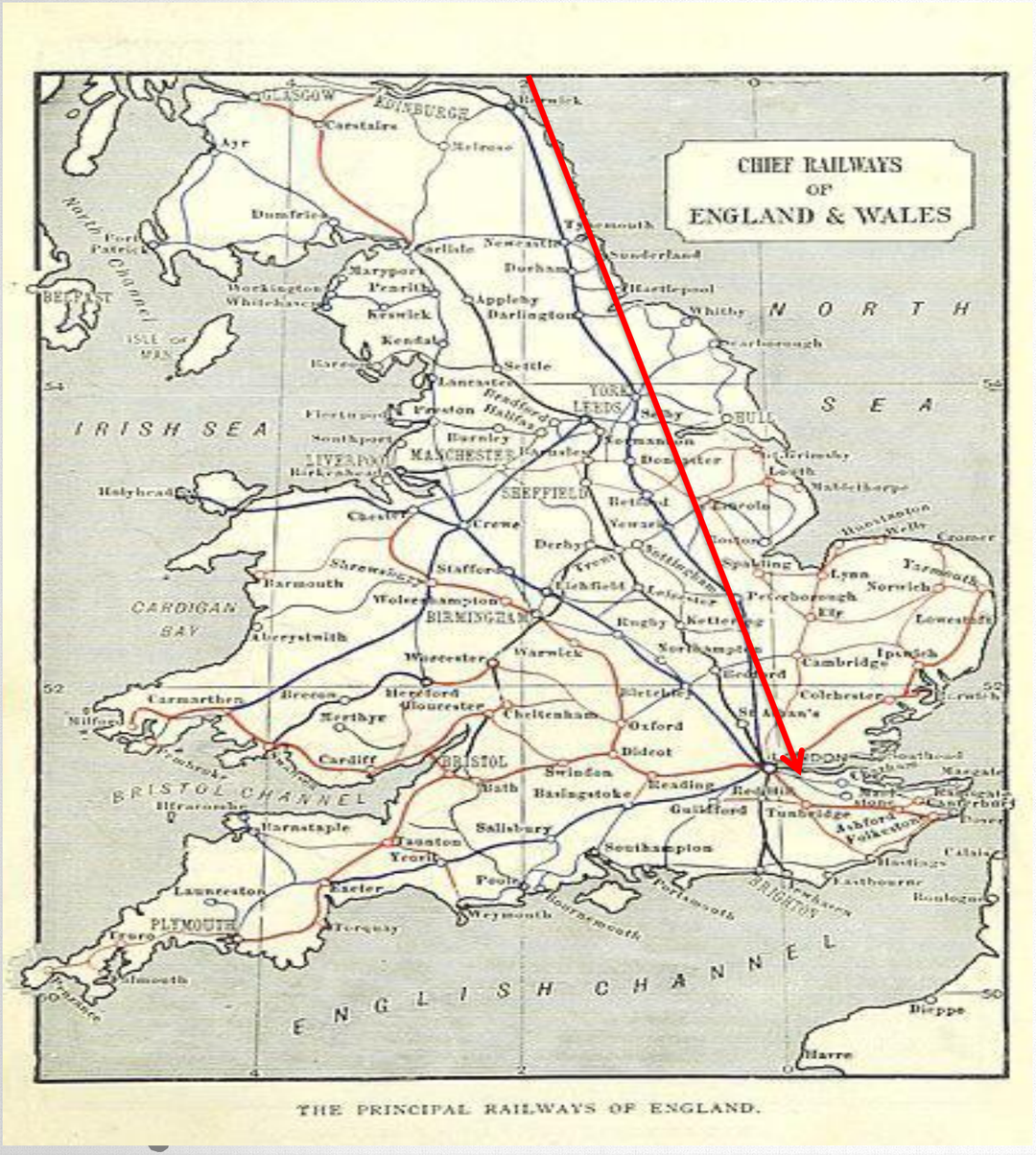
- ¿Qué es un ordenador?
- ¿Qué se puede hacer con un ordenador?

- Los ordenadores pueden realizar gran cantidad de tareas ya que son programables.

CONTROL STRUCTURES

Descanso – volvemos a las 12:00

ONE SINGLE TRAIN (no parallelism) and while and if code

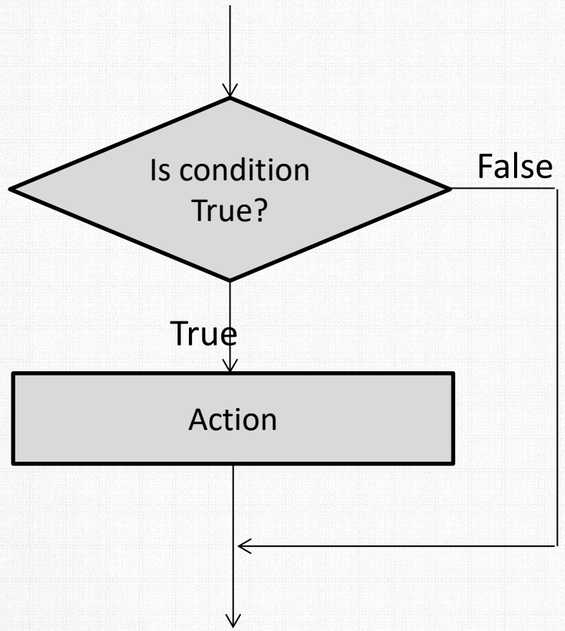


THE PRINCIPAL RAILWAYS OF ENGLAND.

```
#Demonstrates logical operators and compound conditions

print("\t Super advanced security System")
print("\t\t Members Only! \n")

security = 0

username=""
#input in a while loop to force a non empty string inpu
while not username:
    username = input("Please input your username: ")

password = ""
#input in a while loop to force a non empty string inpu
while not password:
    password = input("Please input your password: ")

if username == "Juanjo" and password =="Casado" :
    print("hello boss!!")
    security = 5

elif username == "Bill" and password == "Gates" :
    print("show me the money!!!")
    security = 3

elif username == "guest" or password == "guest" :
    print("I don't know you")
    security =1

else:
    print("Get out of here")
    security =0
```

IF - Selective Control Structures.

IF-WITHOUT ELSE: If the Condition is True, the Action is executed, otherwise the algorithm continues with its execution.



GENERIC SINTAX IN PYTHON

```
if (condition):
  action
```

PYTHON CODE EXAMPLE:

```
x = 5
if(x > 0):
    print("Positive number")
```

R CODE EQUIVALENT:

```
x <- 5
if(x > 0){
    print("Positive number")
}
```

CPU

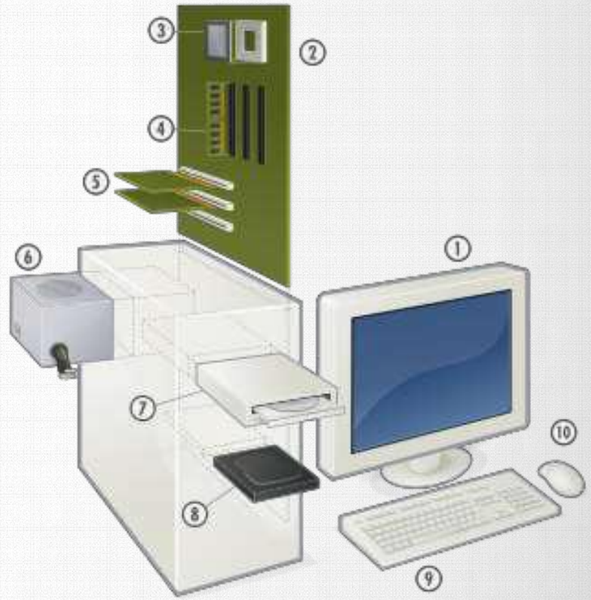
- UCP en castellano. Unidad Central de Proceso
- Es el dispositivo principal del ordenador.
- Es el que ejecuta el programa.
- Interpreta las instrucciones que forman los programas y procesa los datos
- También llamado procesador o microprocesador.

Hardware

- El hardware son los componentes físicos con los que está construido un ordenador.

- Principalmente está formado por:

  - CPU
  - Memoria principal
  - Dispositivos de almacenamiento secundario
  - Dispositivos de entrada
  - Dispositivos de salida



Programación

- ¿Qué es un programa?
- Un programa es un conjunto de instrucciones que un ordenador sigue para poder realizar una tarea.

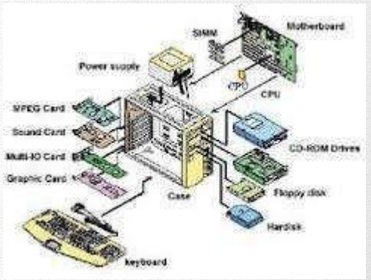
- Flexibilidad.
- Funcionalidad.

Dispositivos de salida

- En inglés output devices.
- Son los dispositivos que permiten al ordenador enviar información al mundo exterior.
- Ej: monitor, impresora, altavoz, etc.

Otros componentes del ordenador

- Tarjeta gráfica
  - Traduce información interna del procesador y memoria a imágenes, posiblemente realizando muchas operaciones
- Tarjeta de sonido
  - Igual que la tarjeta gráfica, pero con sonido. Suelen ser mucho más simples que las gráficas
- Tarjeta de red
  - Comunicación del ordenador con otros ordenadores



Software

- Todo lo que hace el ordenador está bajo el control del software.
- Un ordenador sin software no serviría para nada.
- Se suele dividir en dos tipos:
- Software de sistemas
  - SO, desarrollo, utilidades
- Software de aplicaciones
  - EL resto de programas

DIAGRAMAS DE FLUJO

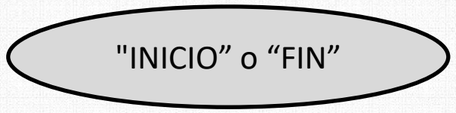
Introducción.

- Es una notación gráfica para implementar algoritmos.
- Se basa en la utilización de unos símbolos gráficos denominados bloques, en los que escribimos las acciones que tiene que realizar el algoritmo.
- Estos bloques están conectados entre sí por líneas y eso nos indica el orden en el que tenemos que ejecutar las acciones.
- En todo algoritmo **siempre** habrá un bloque de inicio y otro de fin, para el principio y final del algoritmo.

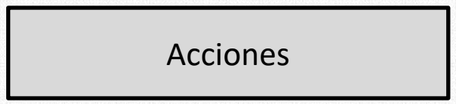
Símbolos Utilizados.

Líneas de flujo: Es una línea con una flecha (y solo una flecha) que permite conectar los bloques del diagrama. La flecha indica la secuencia en la que se van a ejecutar las acciones.

Principio y Fin: Todo algoritmo y por lo tanto, todo diagrama de flujo tiene un principio y un fin (y son únicos). Dentro de este bloque se coloca la palabra “INICIO” o “FIN” según corresponda.



Proceso: Aquí dentro se escribe la acción que debe realizar el programa. Si son varias, se escriben una debajo de la otra, sin olvidarse que se ejecutan una a una en forma secuencial según en qué orden fueron escritas.



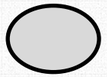
Símbolos Utilizados.

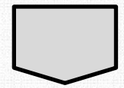
Condición: Dentro de este bloque se escribe una condición. Si ésta es verdadera, entonces el algoritmo tomará una de sus salidas, de lo contrario, tomará la siguiente. Permite representar estructuras del tipo selectivas y repetitivas.

Entrada y Salida: Representa acciones de entrada salida desde un teclado o hacia una pantalla respectivamente. Es decir, si debemos ejecutar una acción que consiste en leer un dato que se ingresa mediante el teclado de una PC y almacenarlo en la variable de nombre “a”, entonces dicha acción se describe dentro de este bloque como “leer a”.

Símbolos Utilizados.

Conectores: Permiten “unir” diagramas de flujo cuando éstos no caben en una misma columna de la hoja por completo. Es decir, cuando debemos, por cuestiones de espacio en la hoja, fragmentar el programa entonces utilizamos estos bloques para indicar los puntos de unión. Cada par de puntos que se deben unir llevarán dentro de este bloque el mismo número.

En la misma hoja: 

En otra hoja: 

Comentarios: Es una aclaración para comprender mejor el código del programa, pero no forma parte del código, es decir, no se ejecuta.

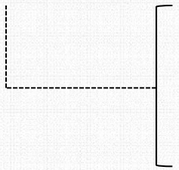
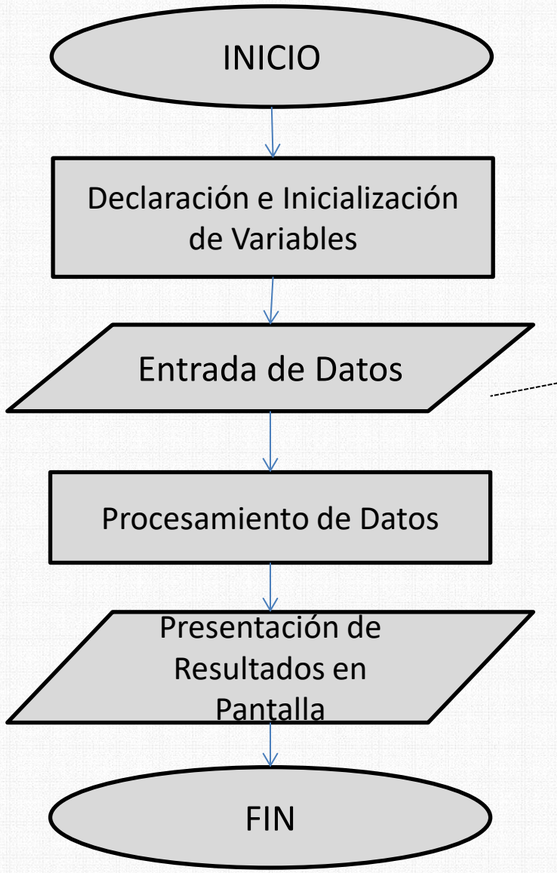


Diagrama de Flujo Elemental.

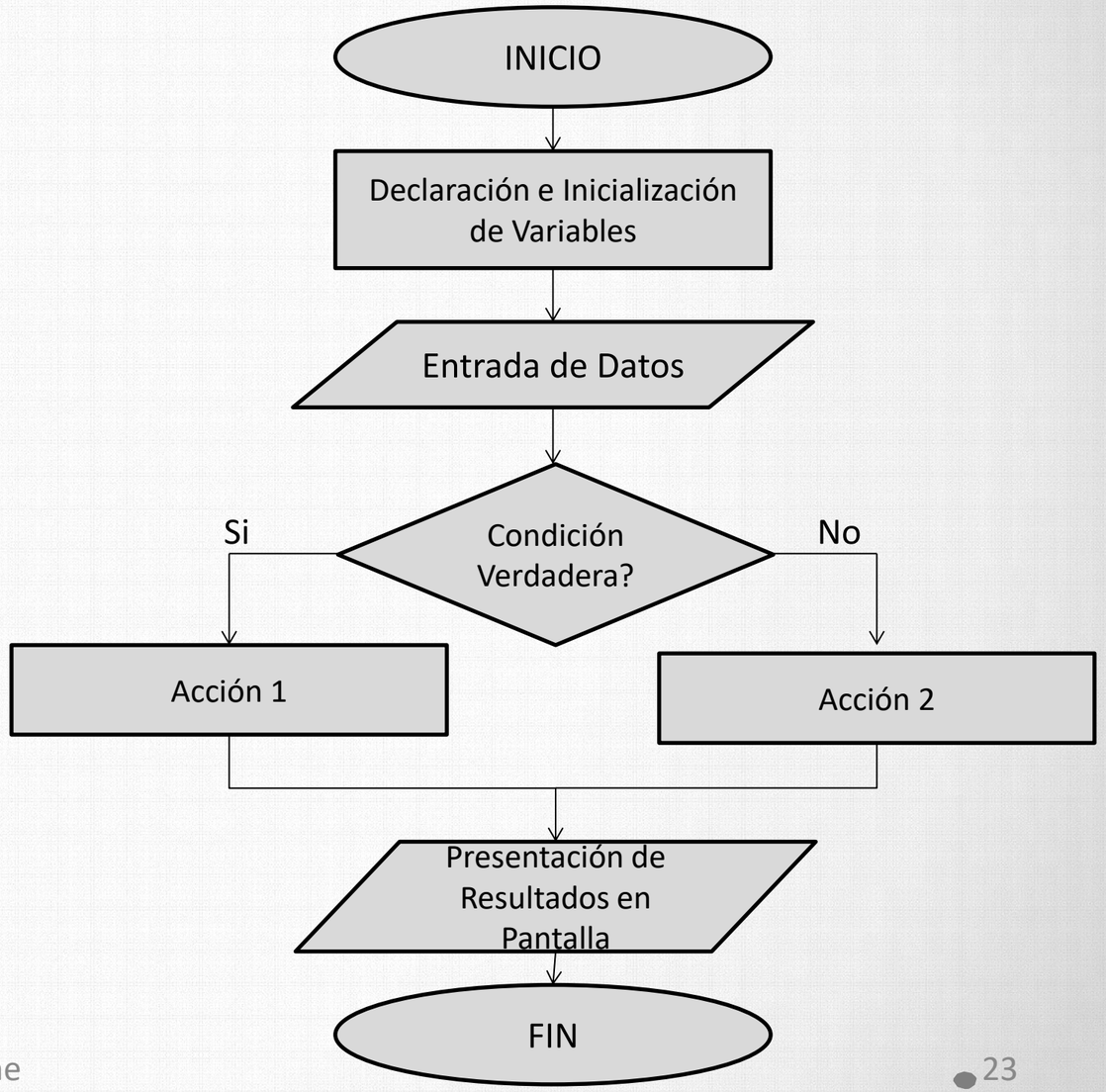


RECORDAR: CONCEPTO DE ALGORITMO: Un algoritmo es un conjunto de pasos o secuencia de instrucciones que, ejecutadas en un determinado orden, permiten resolver un problema determinado.

Diagrama de Flujo Elemental.

Ejemplo de DF de un algoritmo genérico que incluye una bifurcación.

Si la Condición es Verdadera, se ejecuta la Acción 1 y en caso contrario (Falsa) la Acción 2.



PSEUDOCÓDIGO

Pseudocódigo.

- **El pseudocódigo** es una manera de escribir algoritmos de forma poco estricta (con una sintaxis relajada) o estructuras de datos poco detalladas, pero intentando acercar las ideas del algoritmos a estructuras y sintaxis parecidas a las de los lenguajes de alto nivel en los que vamos a programar el algoritmo.

- Es para ser leído por personas, por tanto no se preocupa en detalles sintácticos.

- Es un lenguaje de especificación de algoritmos, pero muy parecido a cualquier lenguaje de programación, por lo que luego su traducción al lenguaje de programación es muy sencillo, pero con la ventaja de que no se rige por las normas de un lenguaje en particular. Nos centramos más en la lógica del problema.

- El pseudocódigo también va a utilizar una serie de palabras claves o palabras especiales que va indicando lo que significa el algoritmo.

Pseudocódigo - Sintaxis Utilizada.

1. **INICIO y FIN:** Por donde empieza y acaba el algoritmo.

2. **DATOS:** Aquí se declaran e inicializan las variables que utilizará el algoritmo.

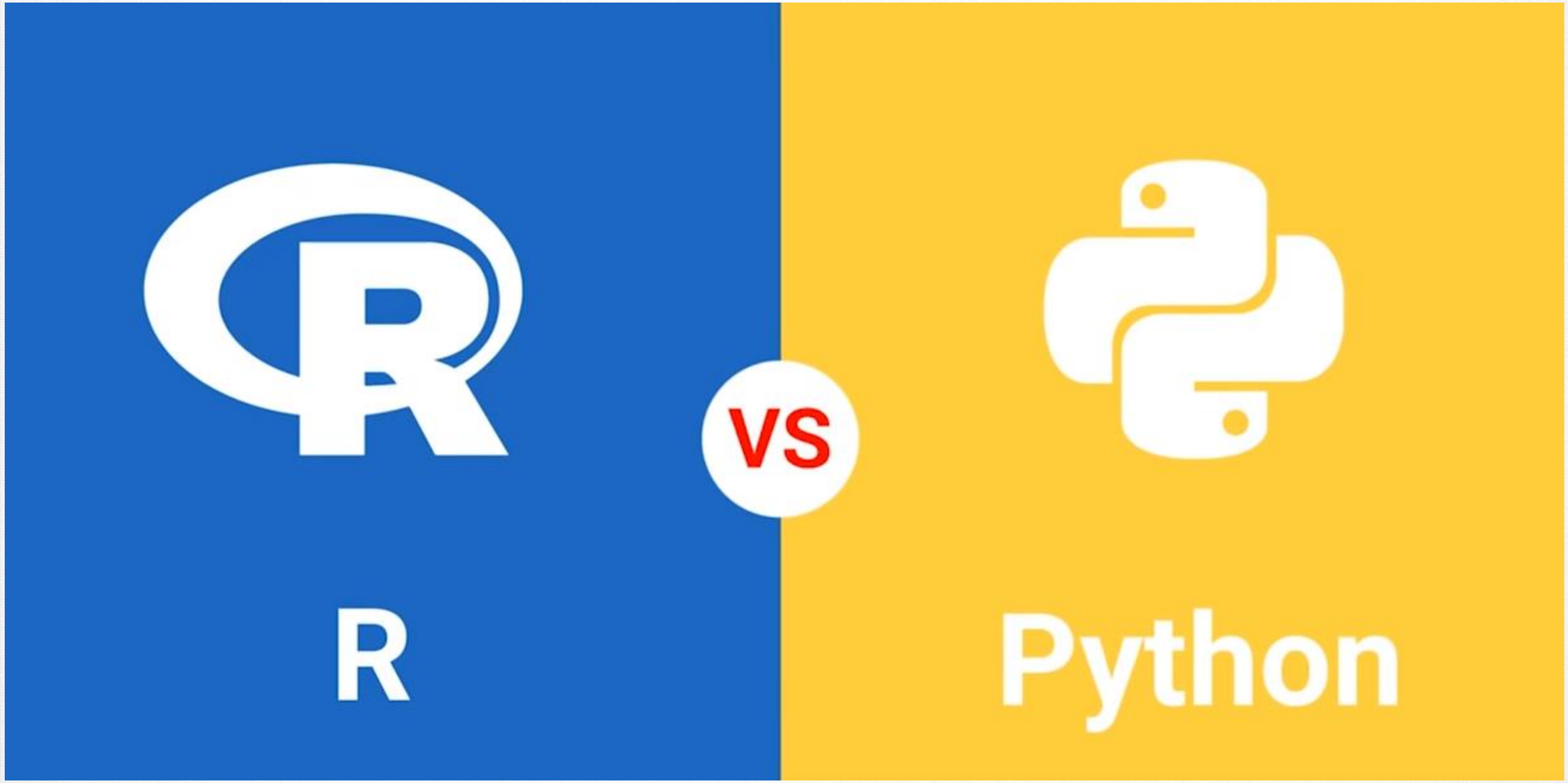
3. **ALGORITMO:** En esta sección se escribe el algoritmo.

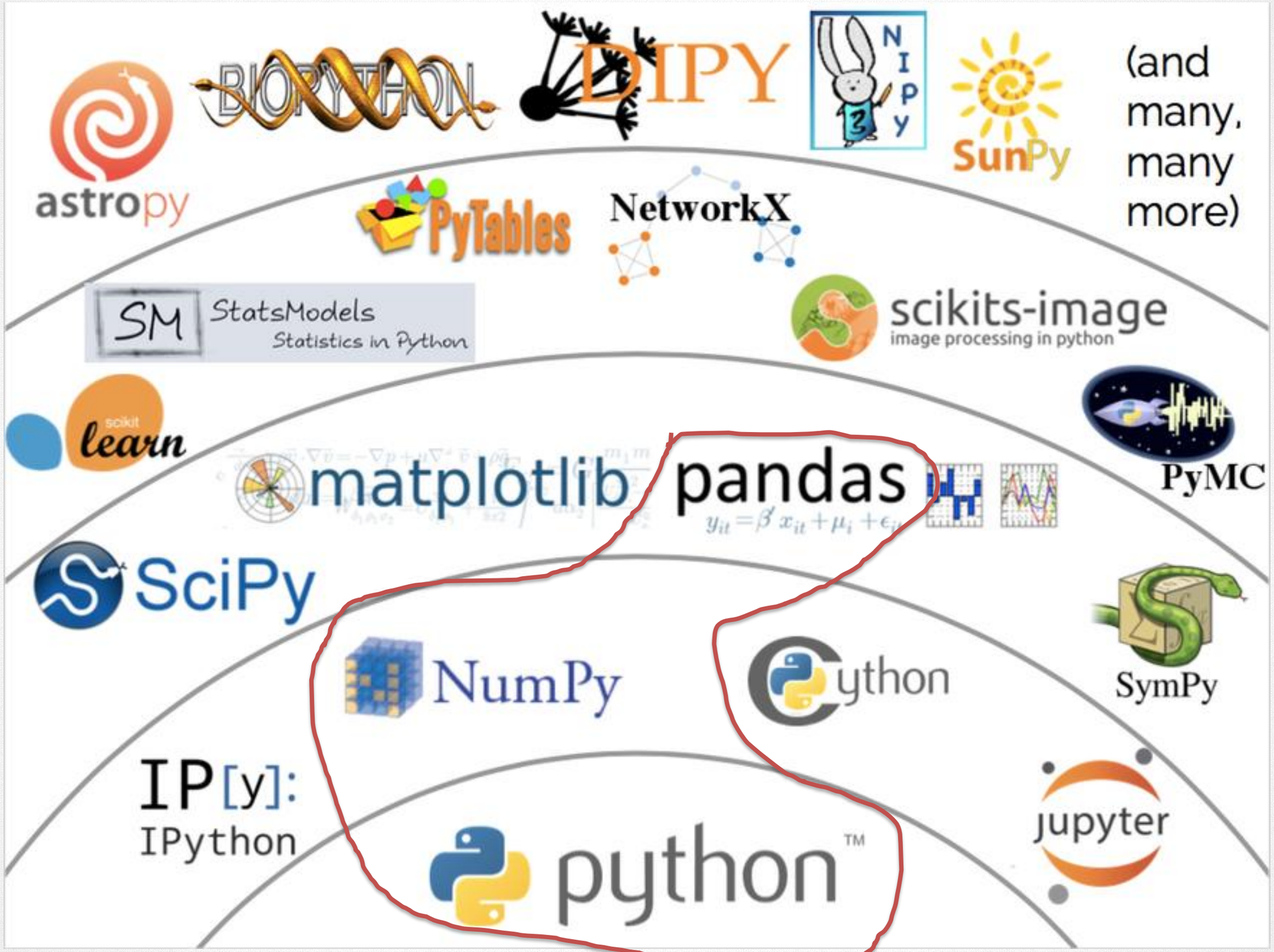
Pseudocódigo de un algoritmo genérico:

```
INICIO.
    DATOS:                      ** esto es un comentario **
        entero a ;                  ** declaración de una variable entera **
        real b = 0 ;        ** declaración e inicialización de una variable **

    ALGORITMO:
        leer a ;
        b = a + 5 ;
        escribir b ;
FIN.
```

R and Python

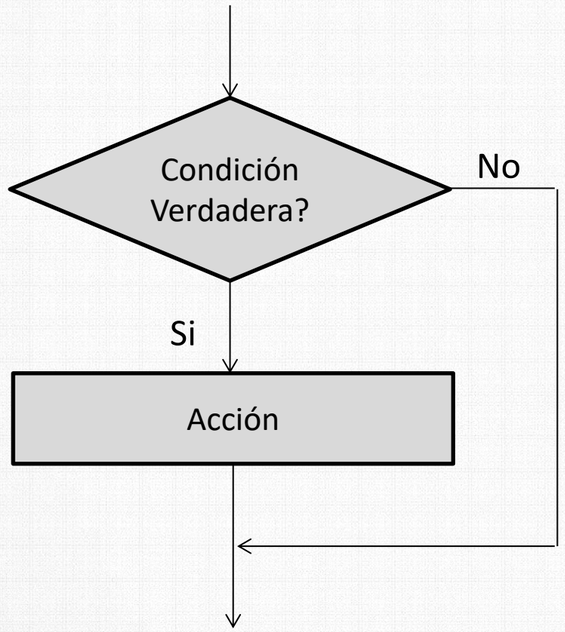
https://www.youtube.com/watch?v=eRP_J2yLjSU




astropy
BIOPYTHON
DIPY
NIPY
SunPy
(and many, many more)
PyTables
NetworkX
SM
StatsModels
Statistics in Python
scikits-image
image processing in python
scikit learn
matplotlib
pandas
PyMC
SciPy
NumPy
Cython
SymPy
IP[y]:
IPython
python
jupyter

Estructuras de Control Selectivas.

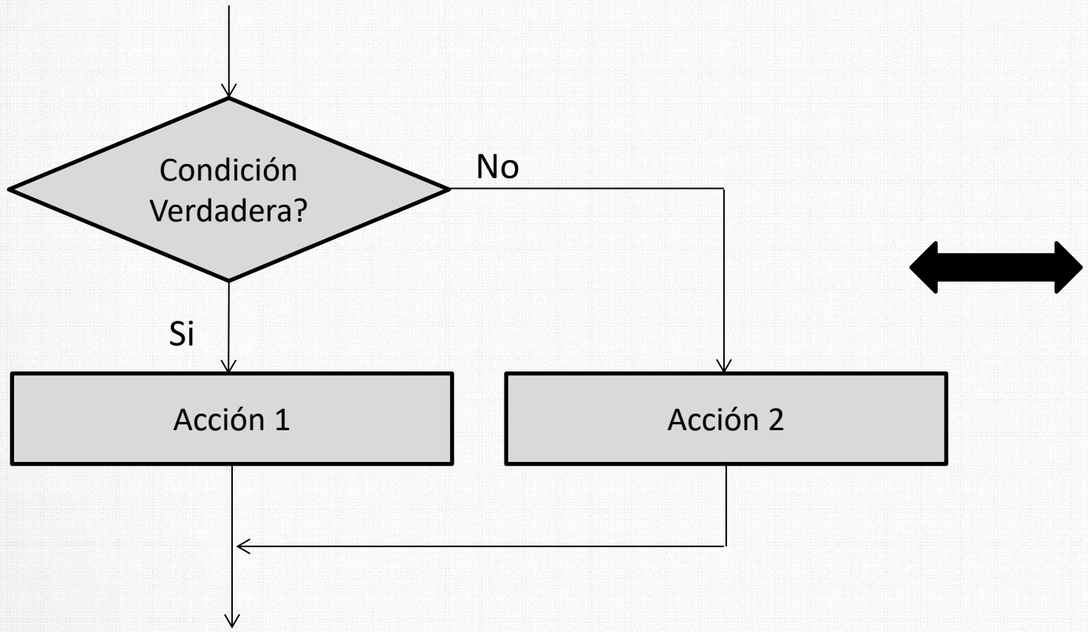
Si: Si la Condición es Verdadera, se ejecuta la Acción, sino el algoritmo continua con su ejecución.



Si < Condición >
    <Acción>
Fin_Si

Estructuras de Control Selectivas.

Si – Sino: Si la Condición es Verdadera, se ejecuta la Acción 1, sino el algoritmo ejecuta la Acción 2. Luego el algoritmo continua con su ejecución.

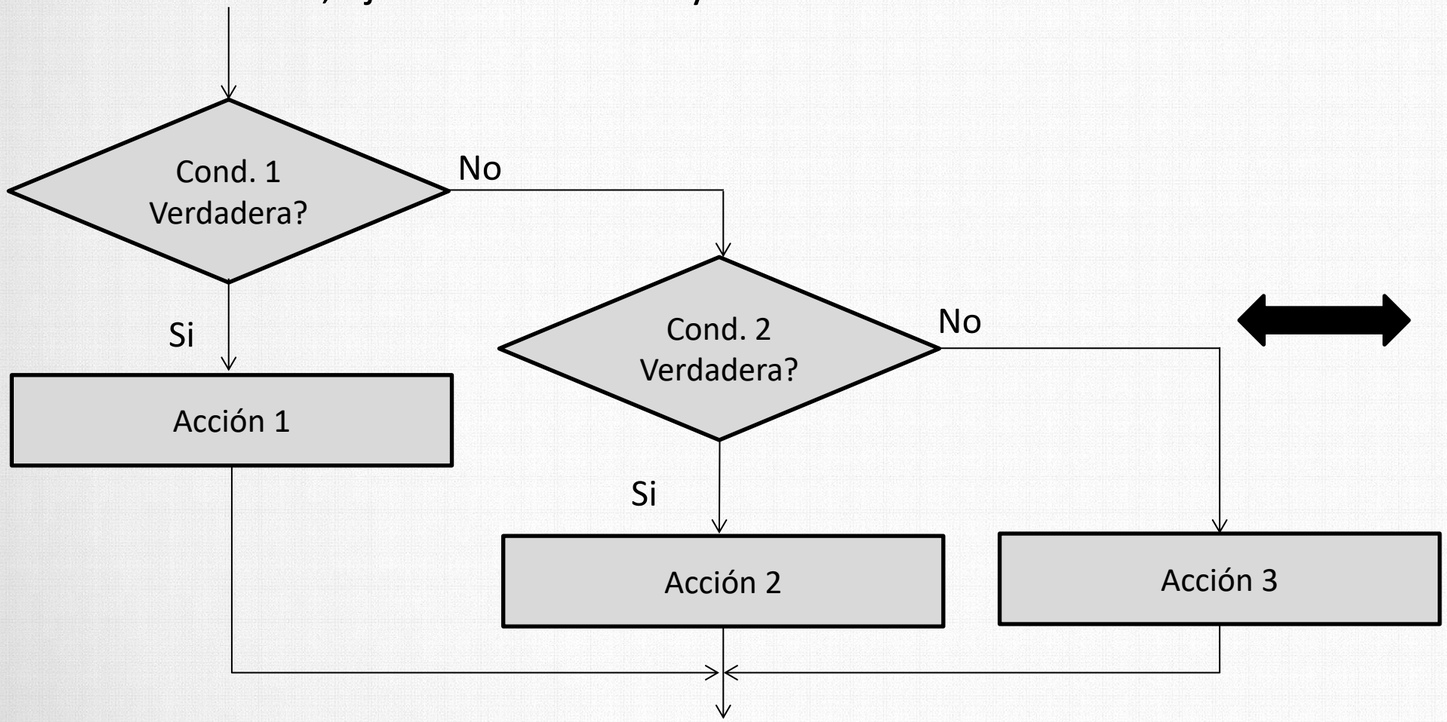


Si < Condición >
< Acción 1 >
Sino < Acción 2 >

Fin_Si

Estructuras de Control Selectivas.

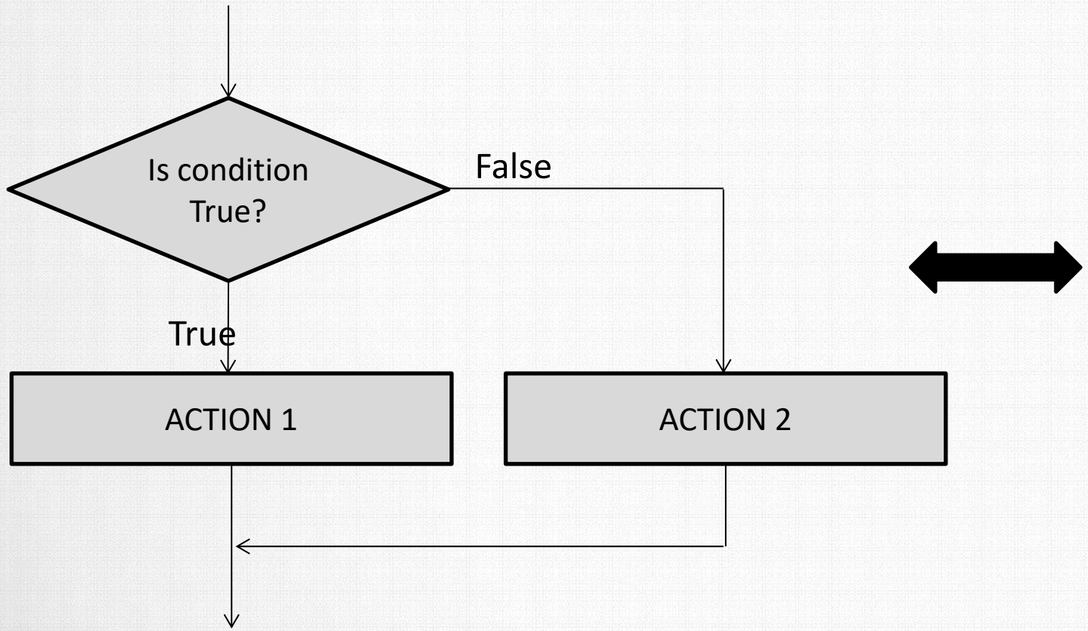
Si – Sino – Si: Si la Condición 1 es Verdadera, se ejecuta la Acción 1, sino el algoritmo evalúa la Condición 2. Si es Verdadera, ejecuta la Acción 2 y sino la Acción 3.




```
Si          < Condición 1 >
                < Acción 1 >
Sino_Si  < Condición 2 >
                < Acción 2 >
Sino          < Acción 3 >

Fin_Si
```

IF-ELSE - Selective Control Structures.

IF-ELSE: If the Condition is True, Action 1 is executed, otherwise the algorithm executes Action 2. Then the algorithm continues with its execution.



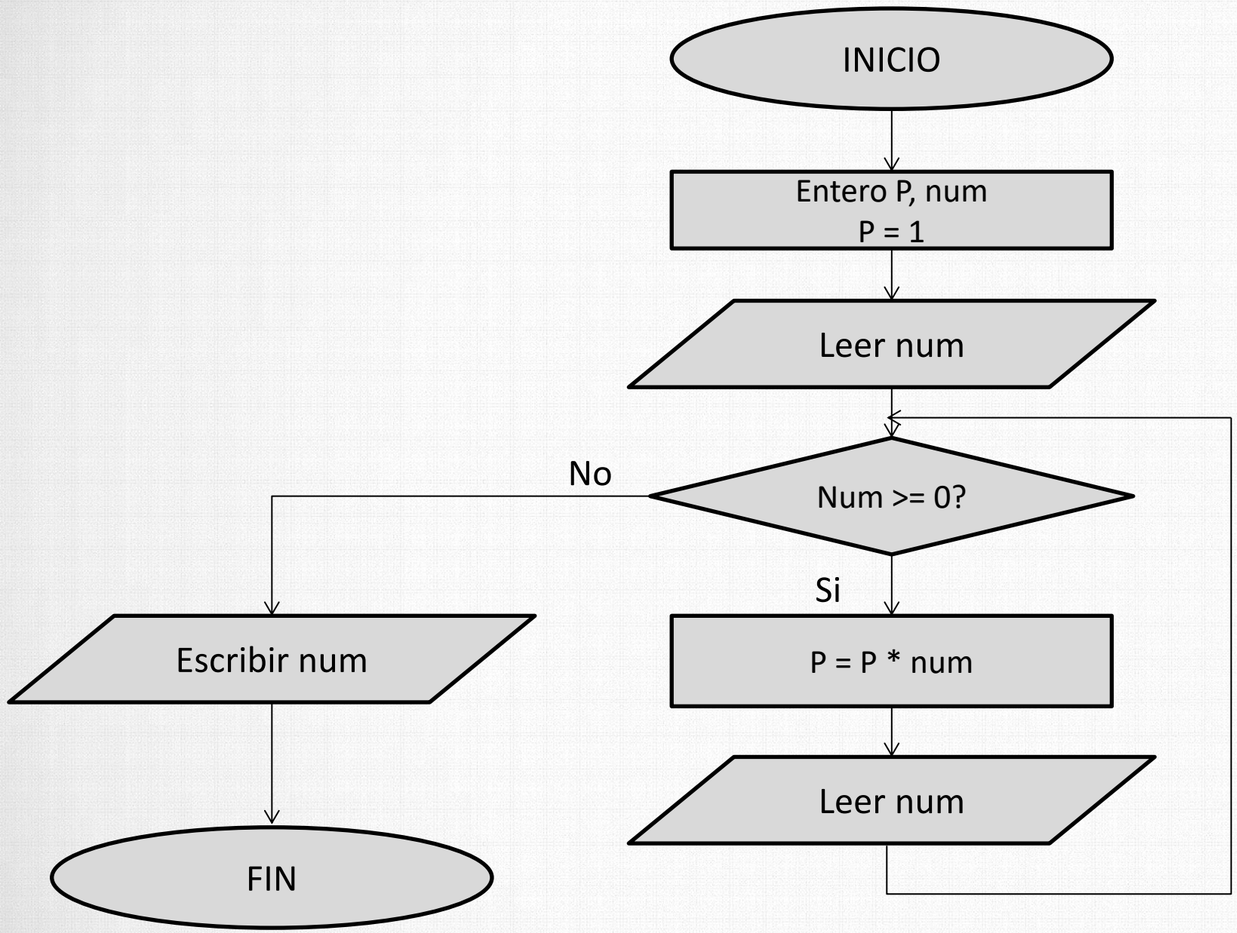

```
GENERIC SINTAX IN PYTHON
if (condition):
    action1
else:
    action2
```

PYTHON CODE EXAMPLE:

```
x = -5
if(x >= 0):
     print("Non-negative number")
else:
     print("Negative number")
```

```
R CODE EXAMPLE EQUIVALENT:
x <- -5
if(x >= 0){
     print("Non-negative number")
} else {
     print("Negative number")
}
```

Ejemplo: Mostrar el producto de números enteros positivos entrados por teclado hasta el ingreso de un número negativo.



Pseudocódigo:

Módulo: Principal

INICIO

DATOS:

P, num: entero

ALGORITMO:

P ← 1

Leer num

Mientras num >= 0

P ← p * num

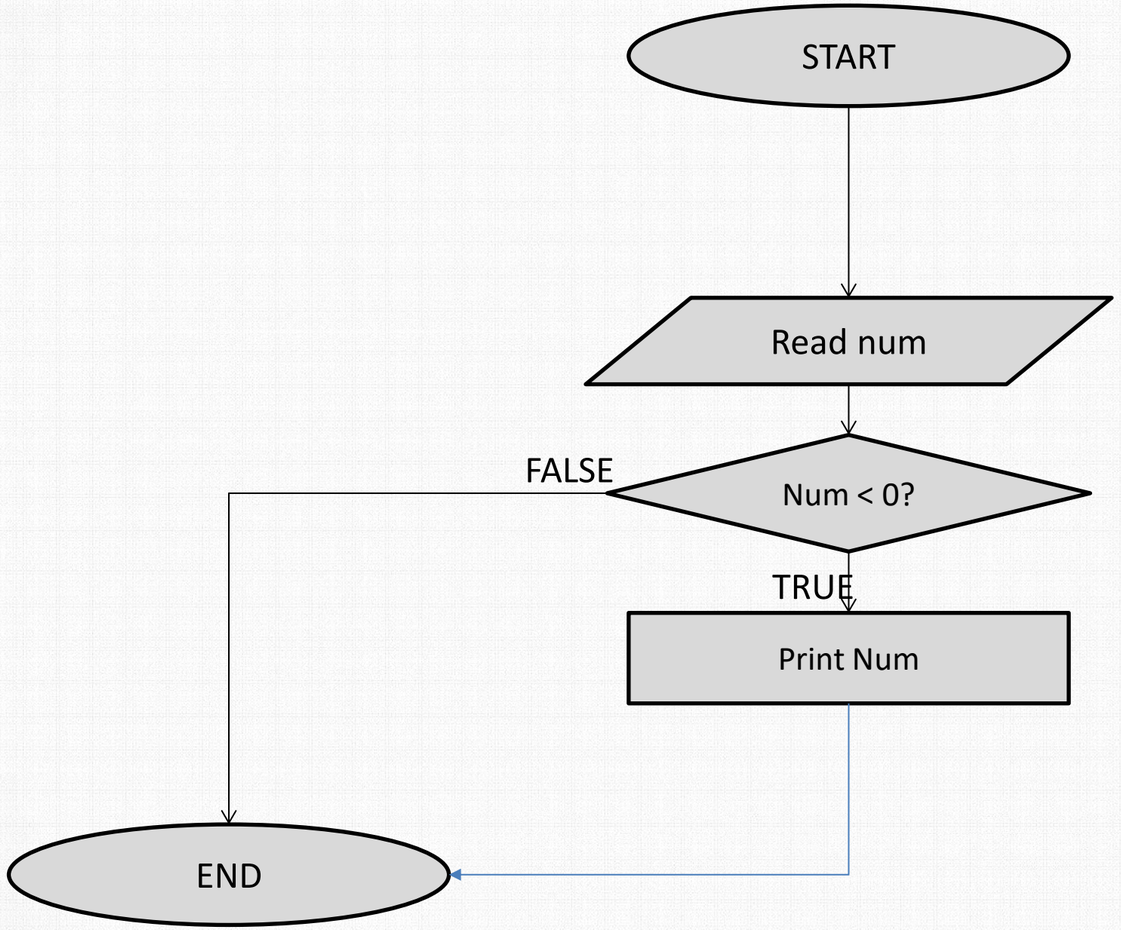
Leer num

Fin_mientras

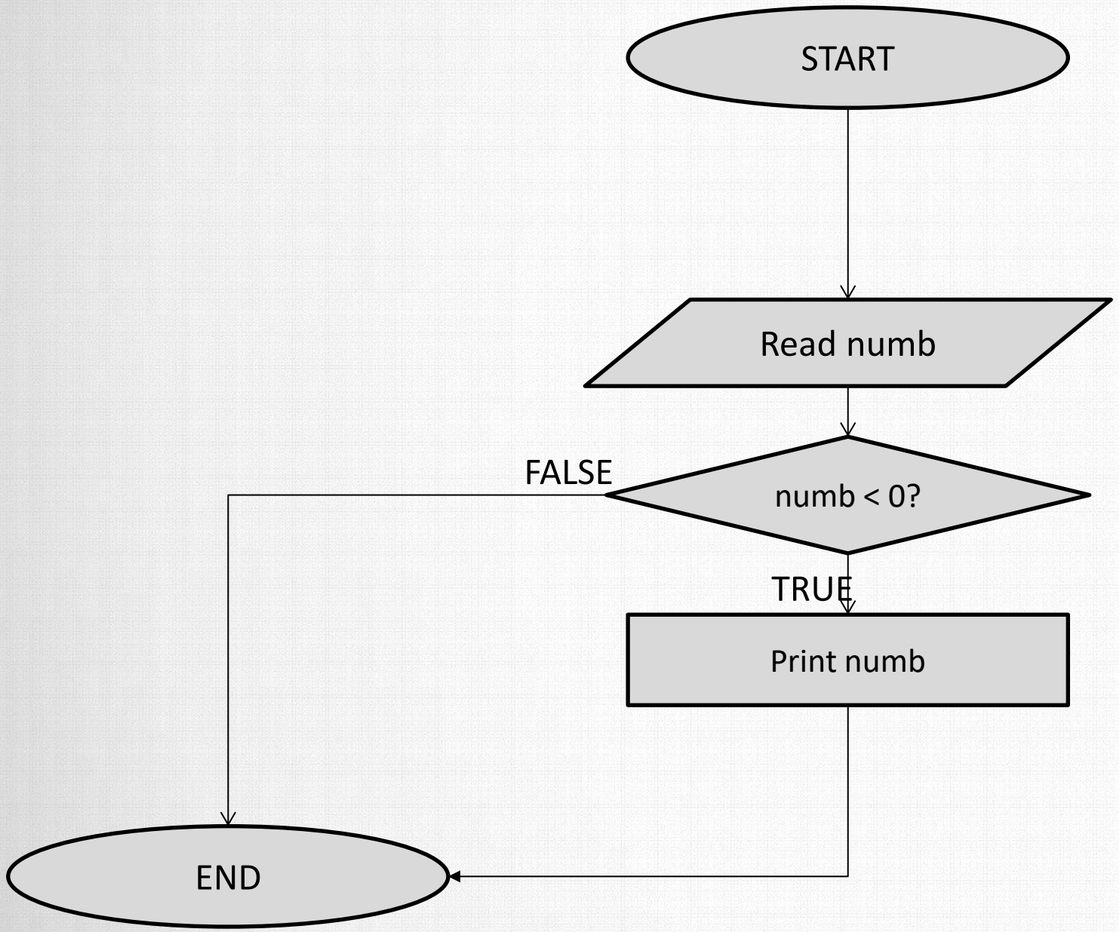
Escribir p

FIN

Exercise – transform the following Flow Diagram into PYTHON code in SPYDER:
Check if the number is negative if it is print the number, otherwise do not print.



Exercise – transform the following Flow Diagram into PYTHON code in SPYDER:
Check if the number is negative if it is print the number, otherwise do not print.



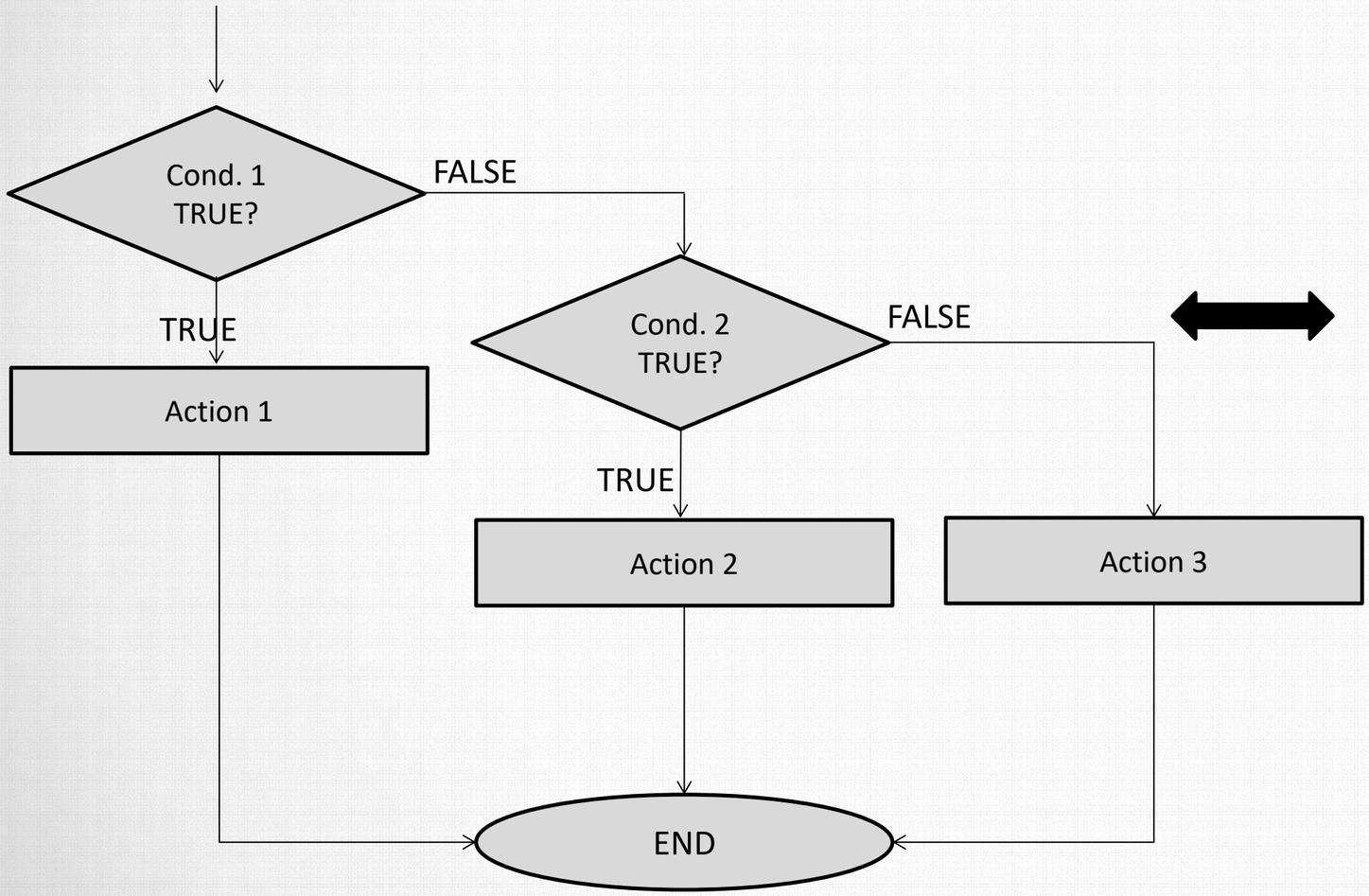
PYTHON CODE SOLUTION:

```
numb = input("Please, enter a number: ")
numb = int(numb)
if (numb < 0):
  print(numb)
```

```
R CODE SOLUTION EQUIVALENT:
numb <- readline(prompt="Please, enter a number: ")
numb <- as.integer(numb)
if (numb < 0) {
  print(numb)
}
```

IF-ELSE-IF - Selective Control Structures.

IF – ELSE – IF: If the condition 1 is TRUE, Action 1 is executed, otherwise the algorithm evaluates if condition 2, if TRUE, action 2 is executed, otherwise action 3 is executed.




```
GENERIC SINTAX IN PYTHON
if (condition 1):
    action 1
elif (condition 2):
    action 2
else:
    action 3
```

PYTHON CODE EXAMPLE:

```
x = -5
if(x > 0):
     print("Positive number")
elif (x < 0):
     print("Negative number")
else:
     print("Zero")
```

```
R CODE EXAMPLE EQUIVALENT:
x <- -5
if(x > 0){
     print("Positive number")
} else if (x < 0) {
     print("Negative number")
} else {
     print("Zero")
}
```

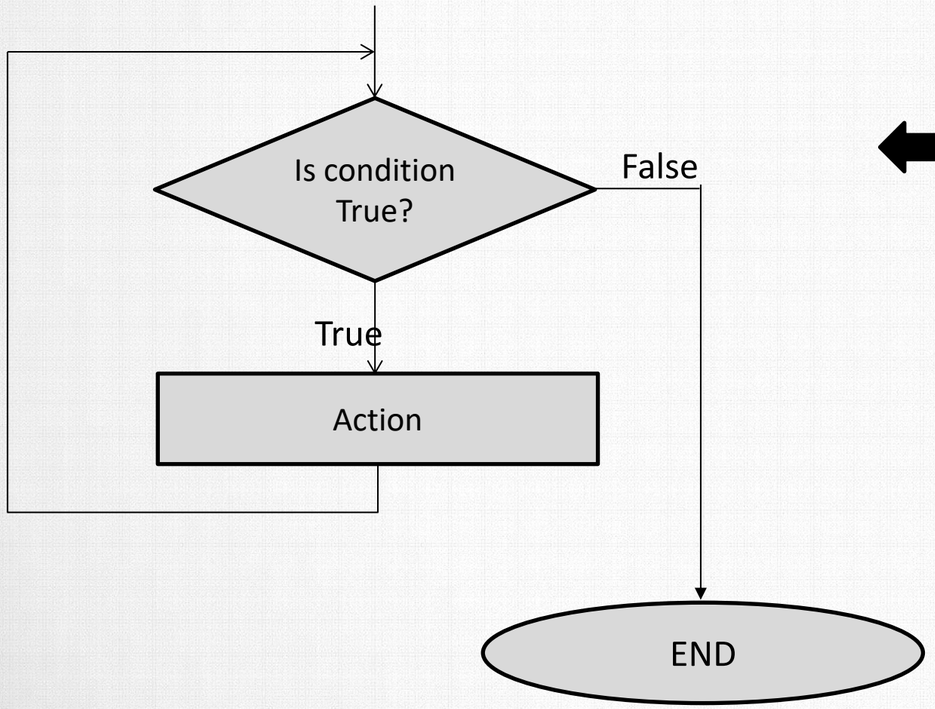
Dispositivos de entrada

- En inglés input devices.
- Son los dispositivos que permiten al ordenador obtener información del mundo exterior.
- Ej: teclado, ratón, USB, CD/DVD/BR, etc.

The Big Bang Theory - The Friendship Algorithm

https://www.youtube.com/watch?v=k0xgjUhEG3U

While: The Condition is evaluated. If True, the Action is executed and the condition is evaluated again. As soon as the Condition is False, the loop is exited and the algorithm continues to run. When the Condition is evaluated, at the beginning, before entering the loop, if the condition is False, it will never enter the loop. Therefore this type of loop is obligatorily used, in the event that, there is a possibility that the loop can be executed 0 times.



GENERIC SINTAX IN PYTHON

```
while (condition):
  action
else:
  final_action
```

PYTHON CODE EXAMPLE:

```
i = 5
while(i > 0):
    print(".")
    i = i - 1
```

R CODE EXAMPLE EQUIVALENT:

```
i <- 5
while(i > 0){
    print(".")
    i <- i - 1
}
```

Python past, present, and future with Guido van Rossum

Episode #100, published Wed, Feb 22, 2017, recorded Wed, Jan 18, 2017.

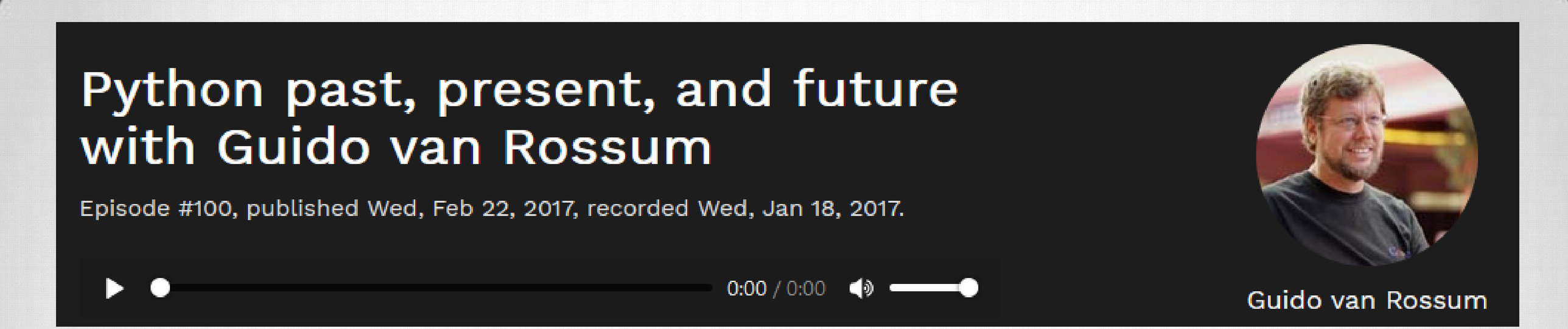
0:00 / 0:00

Guido van Rossum

Play Introduction (up to 0:30 and minute 3:00).
https://talkpython.fm/episodes/show/100/python-past-present-and-future-with-guido-van-rossum

Python past, present, and future with Guido van Rossum

Episode #100, published Wed, Feb 22, 2017, recorded Wed, Jan 18, 2017.



In Guido's first university year - what were the only two languages thought?

A) Python & Java

B) C++ & VBA

C) Algo & Pascal

D) Fortran & Pascal

Answer at 4:32
https://talkpython.fm/episodes/show/100/python-past-present-and-future-with-guido-van-rossum

Python past, present, and future with Guido van Rossum

Episode #100, published Wed, Feb 22, 2017, recorded Wed, Jan 18, 2017.

0:00 / 0:00

Guido van Rossum

In Guido's first university year - what were the only two languages thought?

A) Python & Java

B) C++ & VBA

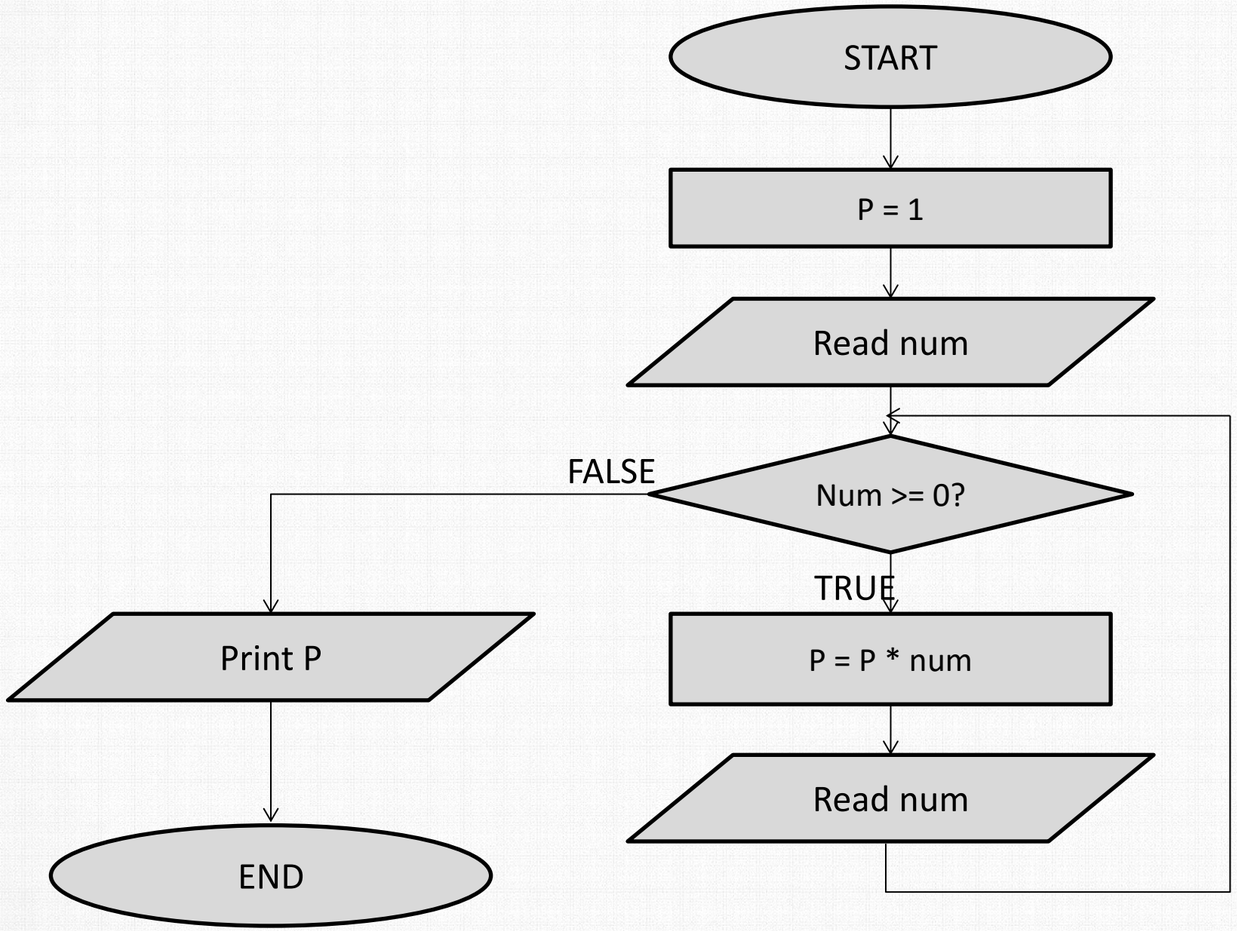
C) Algo & Pascal

D) Fortran & Pascal

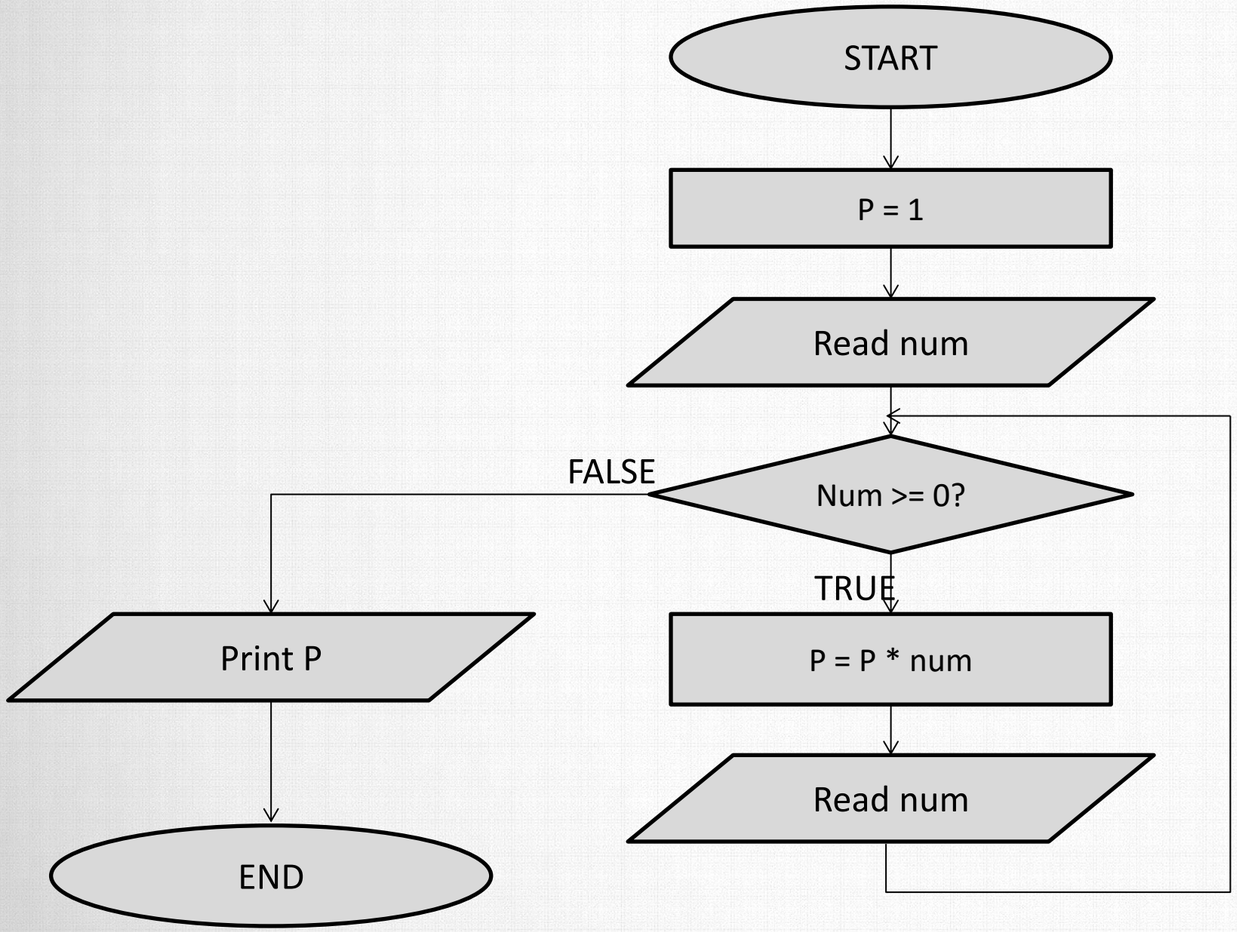
Answer C - 4:32
https://talkpython.fm/episodes/show/100/python-past-present-and-future-with-guido-van-rossum

EJERCICIO + DESCANSO: VOLVEMOS A 14:15

Exercise 2 – transform the following Flow Diagram into PYTHON code: Show the product of positive integers entered by keyboard until the entry of a negative number.



Solution



PYTHON CODE SOLUTION:

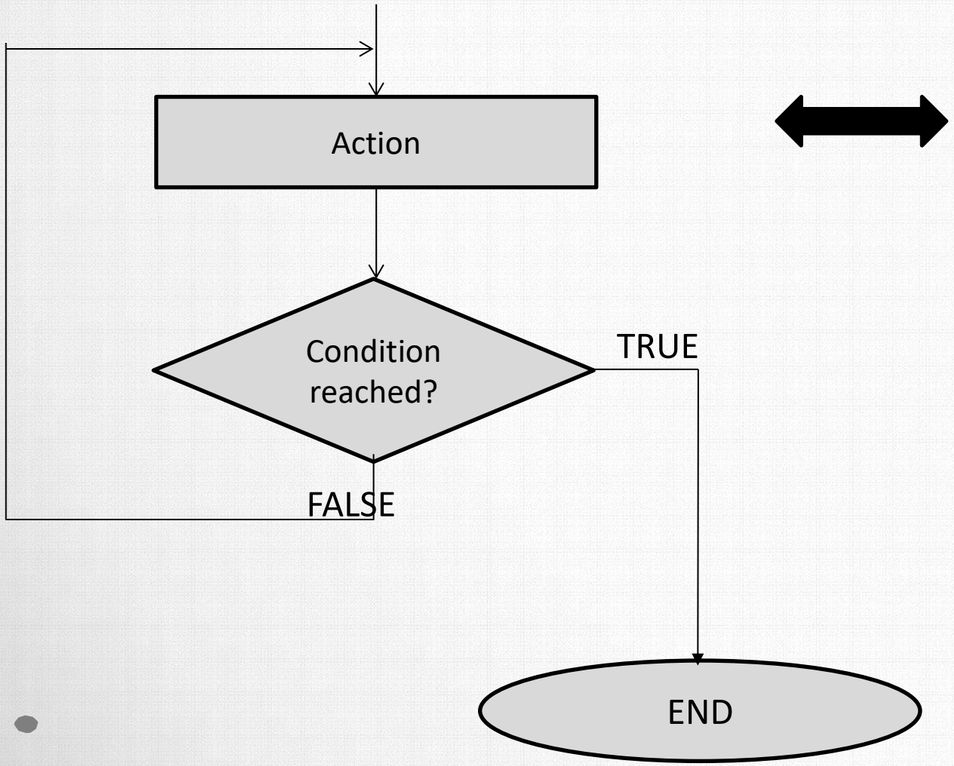
```
P = 1
numb = input("Please, enter a number: ")
numb = int(numb)

while numb>=0:
  P = P * numb
  numb = int(input("Please, enter a number: "))
  numb = int(numb)

print(P)
```

R CODE SOLUTION EQUIVALENT:

```
P <- 1
numb <- readline(prompt="Please, enter a number: ")
numb <- as.integer(numb)

while (numb>=0) {
  P <- P * numb
  numb <- readline(prompt="Please, enter a number: ")
  numb <- as.integer(numb)
}
print(P)
```

REPEAT LOOP - Repetitive Control Structures.

Do – While (or repeat while): In the first instance, the action is executed. Subsequently, the Condition is evaluated. If True, the Action is executed again. If it is False, the execution of the algorithm continues. The loop repeats as long as the condition is True. The Condition is always evaluated at the end of the loop, if it is True the Action is executed again, if it is False it exits the loop. Since the Condition is evaluated at the end, even if the first time is False, the loop (the Action) will have been executed at least once.



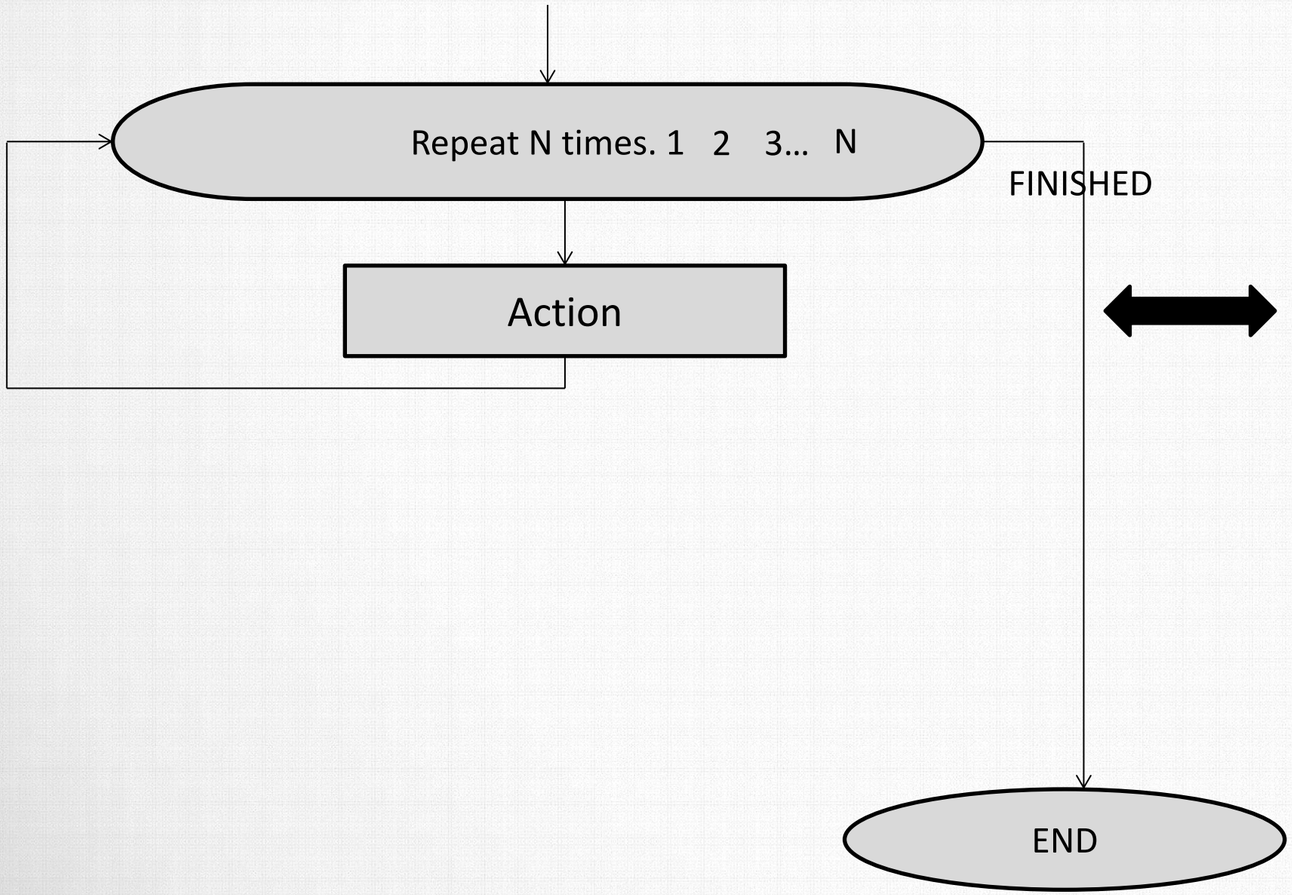

```
GENERIC SINTAX IN R
repeat {
  action
  stops    if    reaches    a
condition
}

R CODE EXAMPLE:

i <- 5
repeat {
  i <- i - 1
  print(".")
  if (i == 0) { break }
}
```

PYTHON DOES NOT HAVE IT!

EMULATING IT IN PYTHON

```
i = 1

while True:
   print(i)
   i = i + 1
   if(i > 5):
      break
```

From - To: Used when the exact number of times to iterate is known. For this, the loop will have an associated variable "index", to which an initial value is assigned and it is determined what its final value will be and it will also increase (or decrease) in each loop iteration by a constant value .



GENERIC SINTAX IN PYTHON

```
for val in sequence:
    statement
```

PYTHON CODE EXAMPLE:

```
x = [2,5,3,9,8,11,6]
count = 0
for val in x: #In python () are not allowed
    if(val % 2 == 0):
        count = count+1

#a blank line is recommended at the end of loops
print(count)
```

R CODE EXAMPLE EQUIVALENT:

```
# Below is an example to count the number of even numbers in a vector.
x <- c(2,5,3,9,8,11,6)
count <- 0
for (val in x) {
   if(val %% 2 == 0) {
      count = count+1
   }
```

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

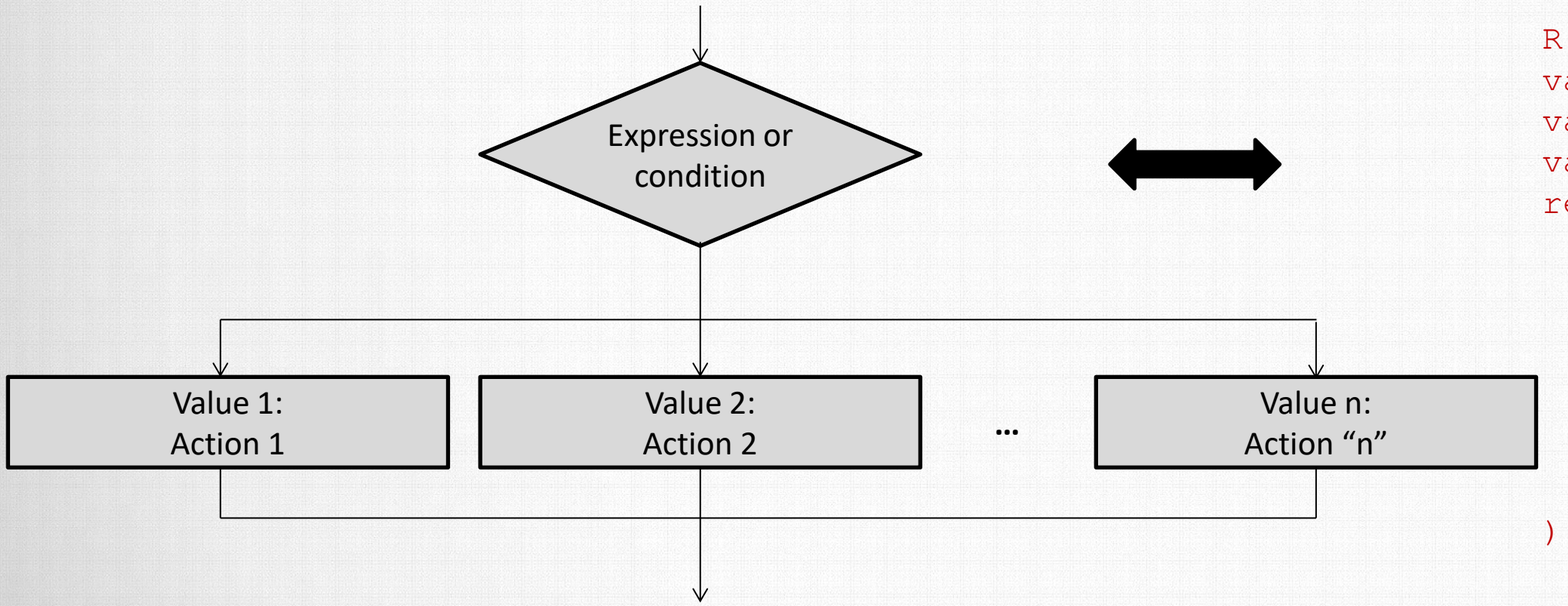
EJERCICIO

- **Escribir un programa que encuentre todos los números que son divisibles por 7 pero que no son múltiplos de 5, entre 2000 y 3200 (ambos incluidos).**

- **Sugerencia: para bucle for y la función range()**

SWITCH - Selective Control Structures.

Switch case (multiple alternative): A condition or expression that can take “n” values is evaluated. According to the value that the expression has at each moment, the actions corresponding to the value are executed. The value with which the expression is compared will depend on the languages, on whatever that value is. In general, that value can be a constant value, a range of values, or even another condition.




```
GENERIC SINTAX IN R
switch(expression, case1, case2, case3....)

R CODE EXAMPLE:
val1 <- 6
val2 <- 7
val3 <- "s"
result <- switch(
  val3,
  "a"= cat("Addition =", val1 + val2),
  "d"= cat("Subtraction =", val1 - val2),
  "r"= cat("Division = ", val1 / val2),
  "s"= cat("Multiplication =", val1 * val2),
  "m"= cat("Modulus =", val1 %% val2),
  "p"= cat("Power =", val1 ^ val2)
)
```

PYTHON DOES NOT HAVE IT! DO IT WITH elif

Python past, present, and future with Guido van Rossum

Episode #100, published Wed, Feb 22, 2017, recorded Wed, Jan 18, 2017.

0:00 / 0:00

Guido van Rossum

How do you consider when something should be in the standard library?

A) Any brand new piece that has been developed and finished.
B) A new API that joins with the current hottest application at the moment.
C) Anything that is being actively developed.
D) Something that is useful for multiple range (a wide variety) of applications.

Right answer at 24:32.
https://talkpython.fm/episodes/show/100/python-past-present-and-future-with-guido-van-rossum

Python past, present, and future with Guido van Rossum

Episode #100, published Wed, Feb 22, 2017, recorded Wed, Jan 18, 2017.

0:00 / 0:00

Guido van Rossum

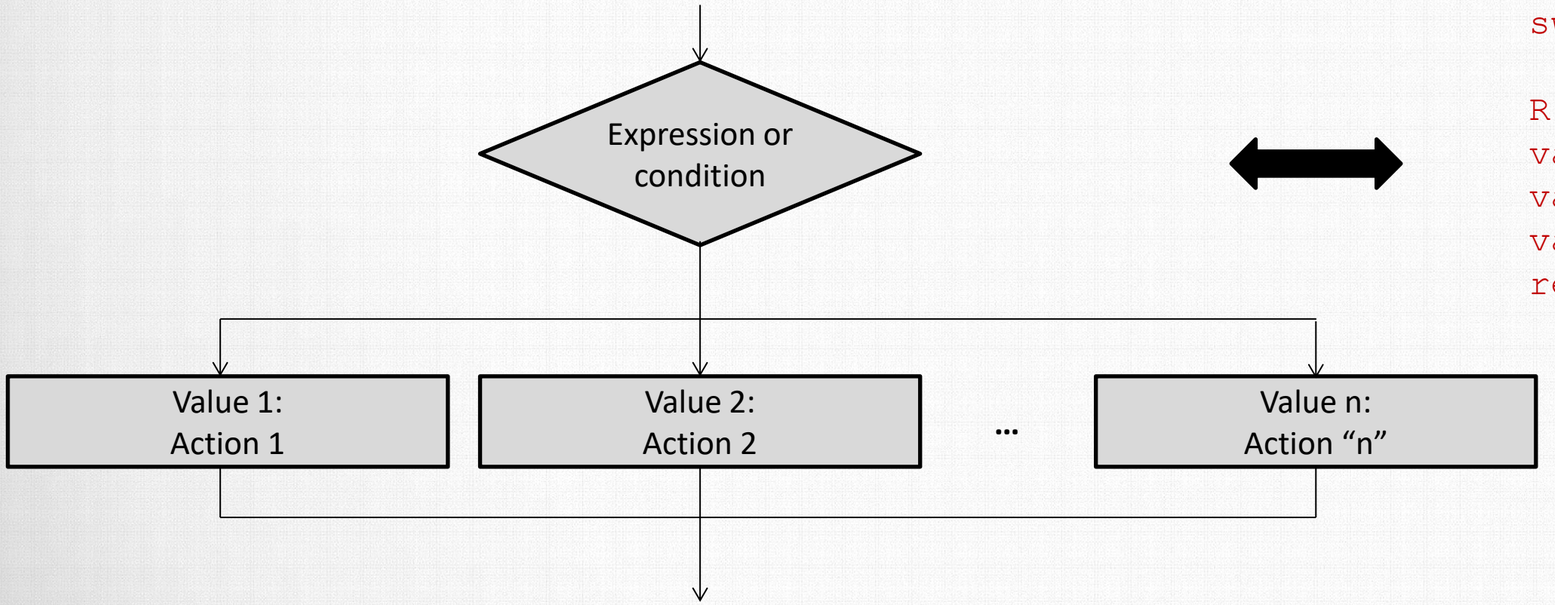
How do you consider when something should be in the standard library?

A) Any brand new piece that has been developed and finished.
B) A new API that joins with the current hottest application at the moment.
C) Anything that is being actively developed.
D) Something that is useful for multiple range (a wide variety) of applications.

Right answer: d) Something that is useful for multiple range (a wide variety) of applications. 24:32.
https://talkpython.fm/episodes/show/100/python-past-present-and-future-with-guido-van-rossum

SWITCH - Selective Control Structures.

Switch case (multiple alternative): A condition or expression that can take “n” values is evaluated. According to the value that the expression has at each moment, the actions corresponding to the value are executed. The value with which the expression is compared will depend on the languages, on whatever that value is. In general, that value can be a constant value, a range of values, or even another condition.




```
GENERIC SINTAX IN R
switch(expression, case1, case2, case3....)

R CODE EXAMPLE:
val1 <- 6
val2 <- 7
val3 <- "s"
result <- switch(
  val3,
  "a"= cat("Addition =", val1 + val2),
  "d"= cat("Subtraction =", val1 - val2),
  "r"= cat("Division = ", val1 / val2),
  "s"= cat("Multiplication =", val1 * val2),
  "m"= cat("Modulus =", val1 %% val2),
  "p"= cat("Power =", val1 ^ val2)
)
```

peppermoney

Kahoot!

Algoritmos y diagramas de Flujo

Player vs Player
1:1 Devices

Classic

Team vs Team
Shared Devices

Team mode

PRE-WORK (hasta 3/Junio)

https://codecombat.com/students?_cc=EnemyDrawGame

Kahoot!

https://kahoot.it/challenge/07614228?challenge-id=a3613dad-8cfd-48a4-b5f2-2043399b46ec_1621944326326

O usar: https://kahoot.it/ y entrar
Game PIN: **07614228**

EJERCICIOS – 11:40

Exercises:

Using women = [43.5, 54.3,68.9,70,2] and only for loops calculate the:

- **Sum**

  Solution:

```
sum = 0
for h in women:
  sum = sum + h
print(sum)
```

- **Min**
- **Max**
- **Average**

$$\bar{x} = \frac{(x_1 + x_2 + \ldots + x_n)}{N} \qquad \bar{x} = \frac{\left(\sum_{k=1}^{n} x_k\right)}{N}$$

- **Standard D**

$$Standard.\ Deviation = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

pwc

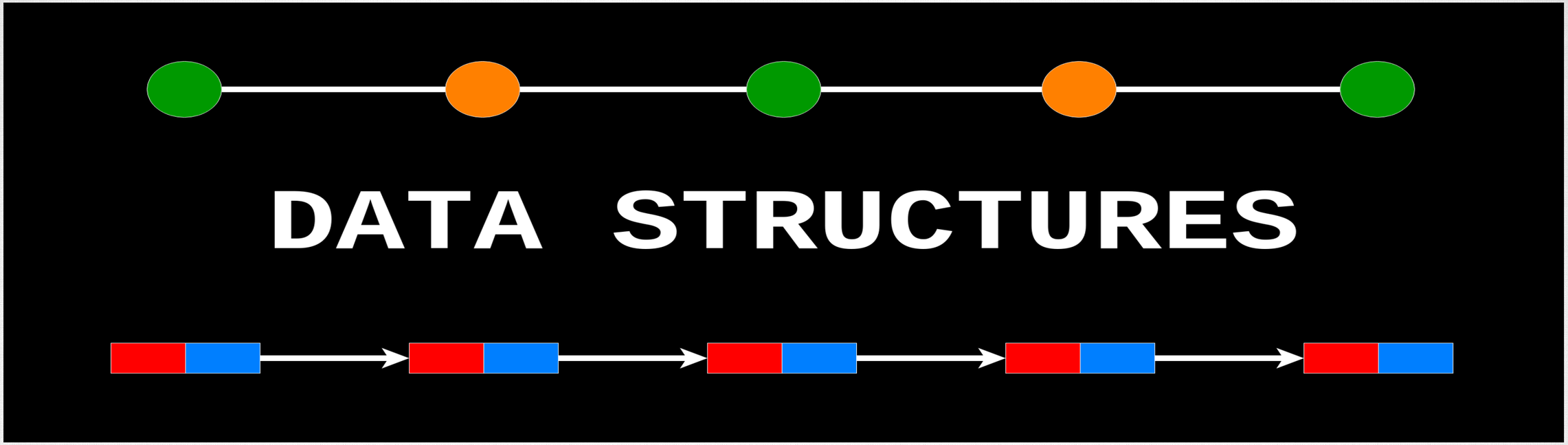
Universidad de Navarra

ESTRUCTURAS DE DATOS

Logical operators

Operator	Description
a > b	Greater than
a < b	Less than
a >= b	Greater than or equal to
a <= b	Less than or equal to
a == b	Check if value of two operands is equal
a != b	Check if value of two operands is not equal

Operator	Description
a **and** b	Logical AND If both operands are True than it returns True
a **or** b	Logical OR If one of the operands is True then it returns True
not	Logical NOT


DATA STRUCTURES

SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

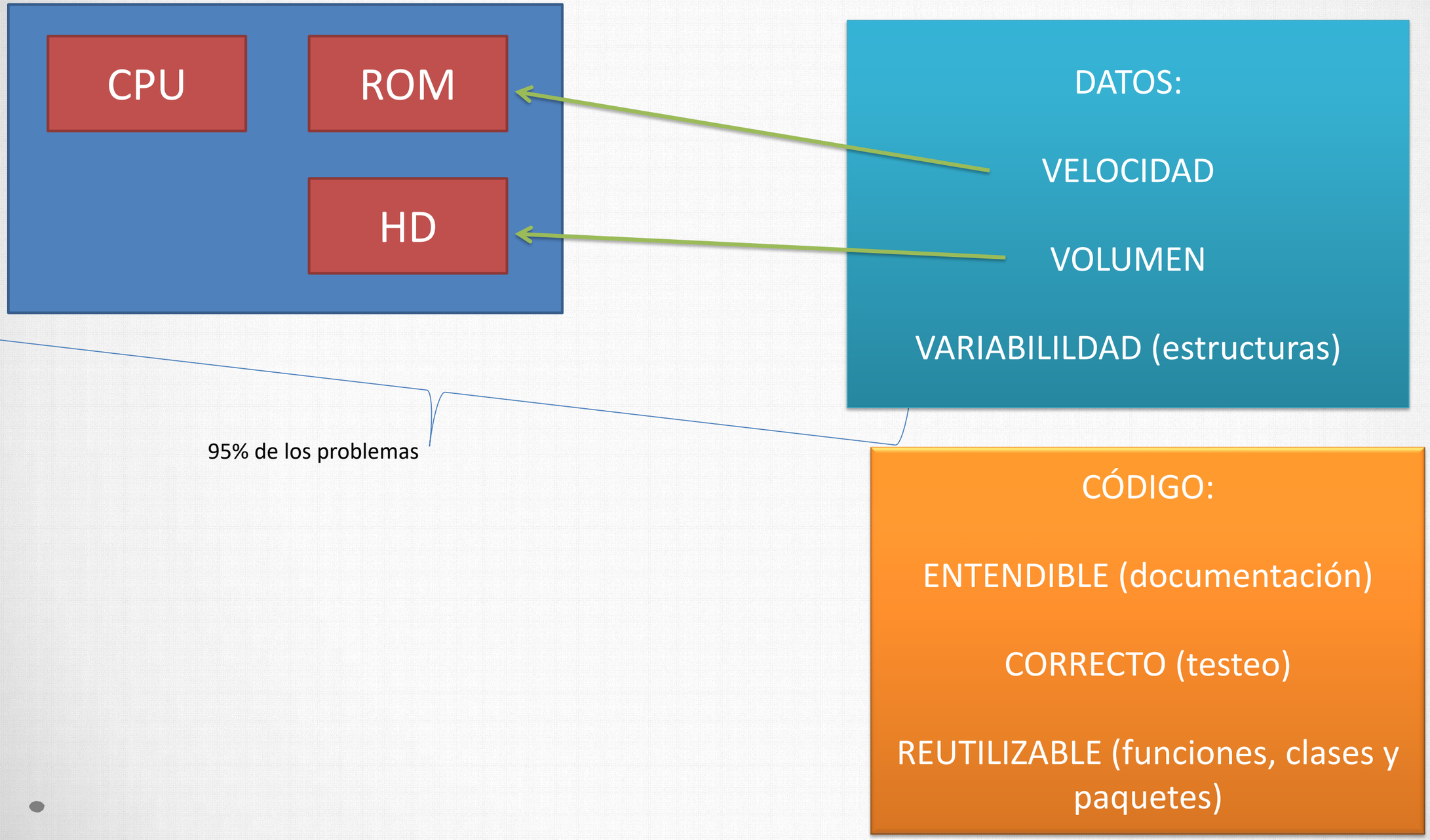
Lists, tuples and Dictionaries

Standard Data Types:

Python has 6 primitive (scalar) data types:
None
Boolean
Integer
Float
Complex
String

What is a structure?

A way of organizing one or more scalar types (none, boolean, integer, float, complex and string).


CPU
ROM
HD
DATOS:
VELOCIDAD
VOLUMEN
VARIABILILDAD (estructuras)
95% de los problemas
CÓDIGO:
ENTENDIBLE (documentación)
CORRECTO (testeo)
REUTILIZABLE (funciones, clases y paquetes)

Memoria principal

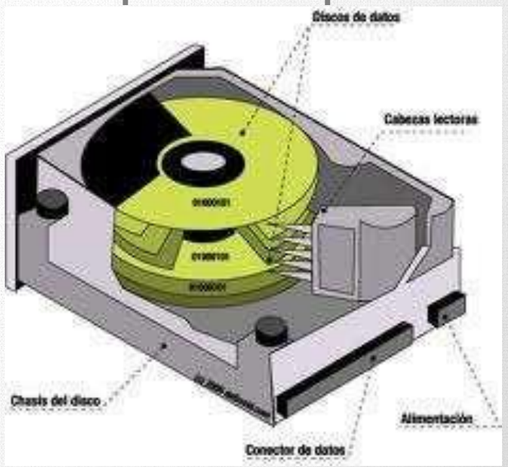
- Main memory en inglés.
- Es donde se almacenas los programas mientras se están ejecutando.
- También se almacenan los datos. (Von Newman)
- RAM: Random Access Memory
  - Memoria de acceso aleatorio
- Se borra cuando se apaga el ordenador
- Es muy rápida

Memoria principal

- La memoria está dividida en pequeñas unidades llamadas bytes
- Los bytes están formados por bits.
- El bit podría definirse como la unidad mínima de información.
- A cada byte en la memoria se le asocia una dirección.

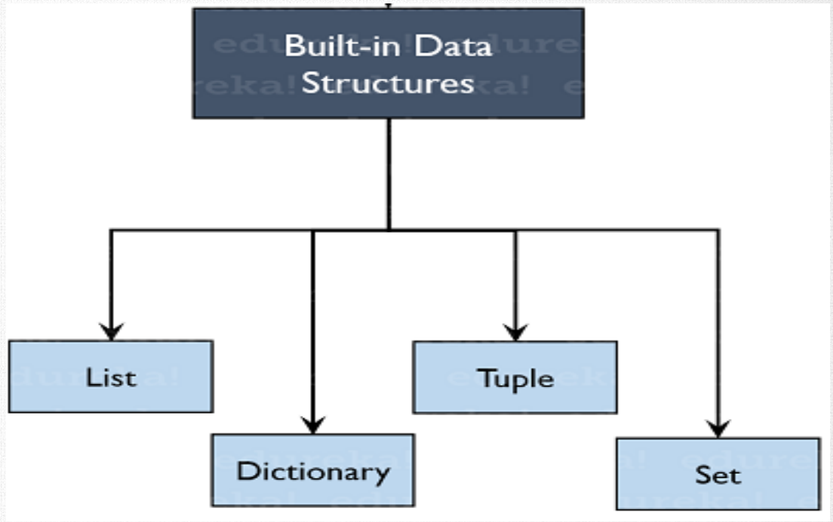
Almacenamiento secundario

- Es un tipo de memoria que almacena datos durante mucho tiempo.
- No se borran cuando se apaga el ordenador.
- Es más lenta que la memoria principal
- Frecuentemente, los programas son almacenados en la memoria secundaria y cargados en la memoria principal cuando es necesario.
- Ej: Discos duros, SSD, etc.



Built-in Data Structures:

Python has 4 standard data structures:
List
Tuple
Dictionary
Set



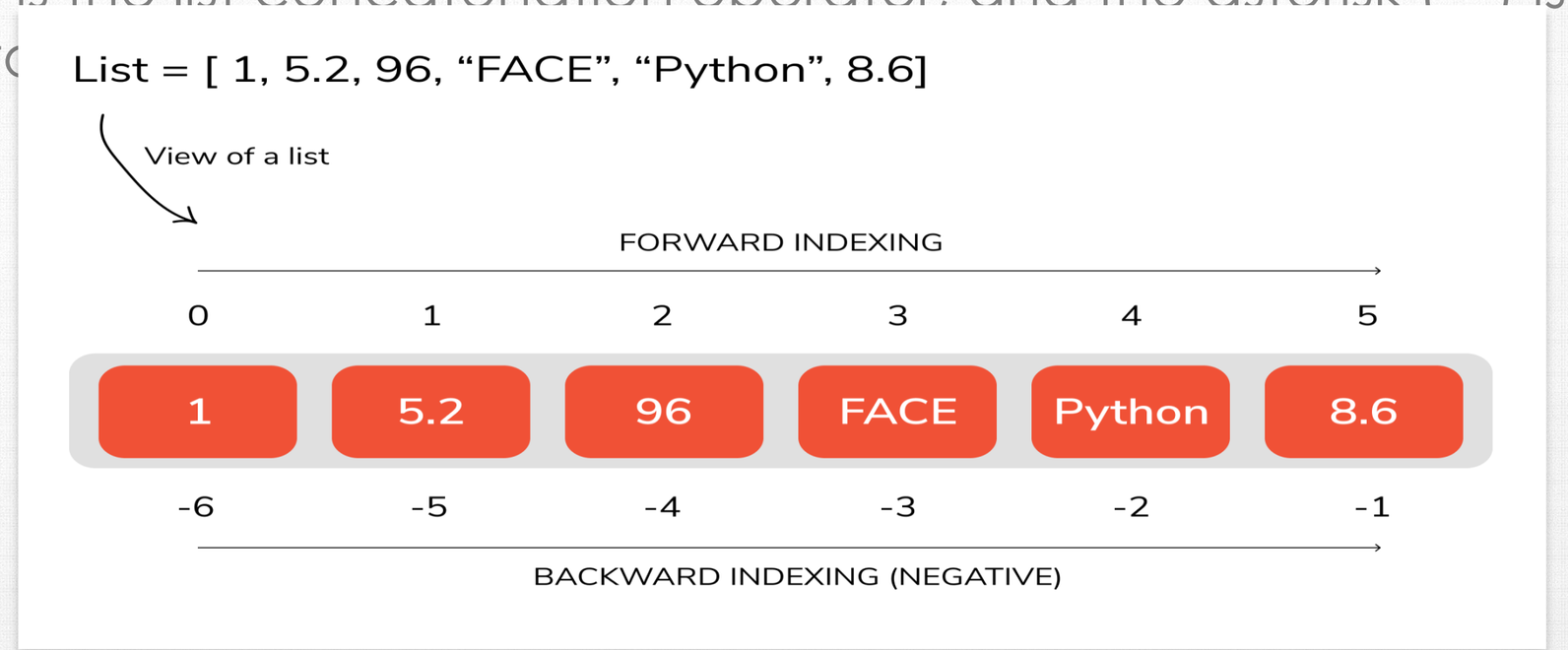
What is a structure?

A way of organizing one or more scalar types (none, boolean, integer, float, complex and string).

SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

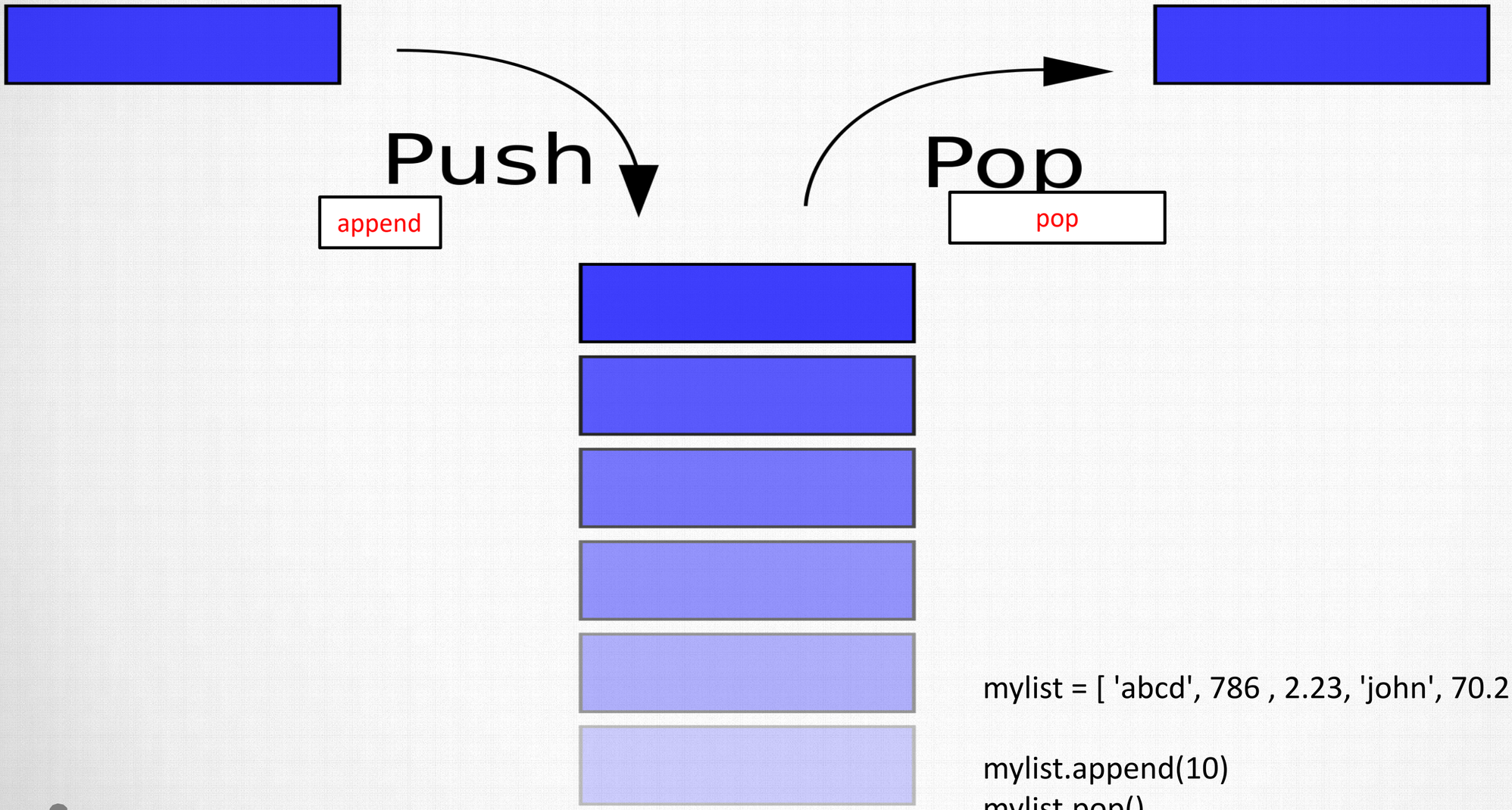
Python Lists:

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).
To some extent, lists are similar to lists in R. However lists are the default option in Python. All the items belonging to a list can be of different data type.
The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end-1.
The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operato



SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

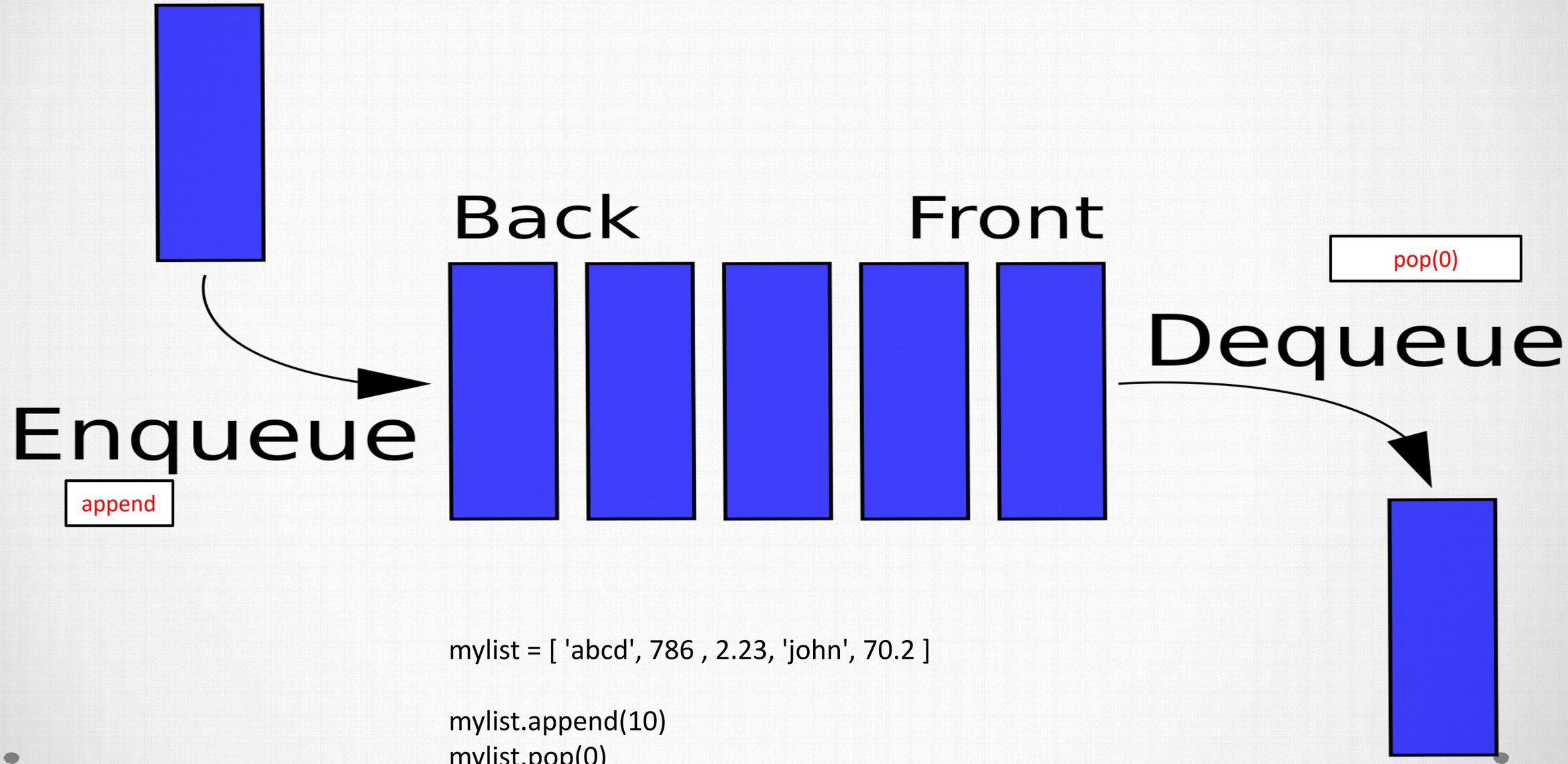
List as stack (Last In First Out)




```
mylist = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

mylist.append(10)
mylist.pop()
```

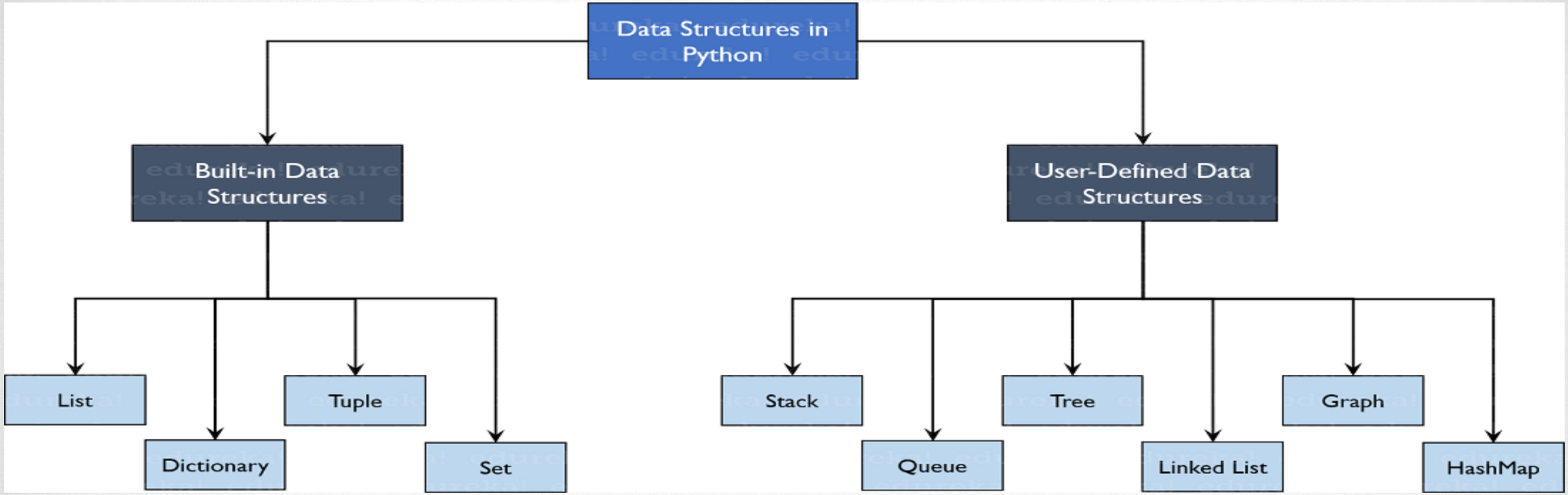
SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

List as queue (First In First Out)

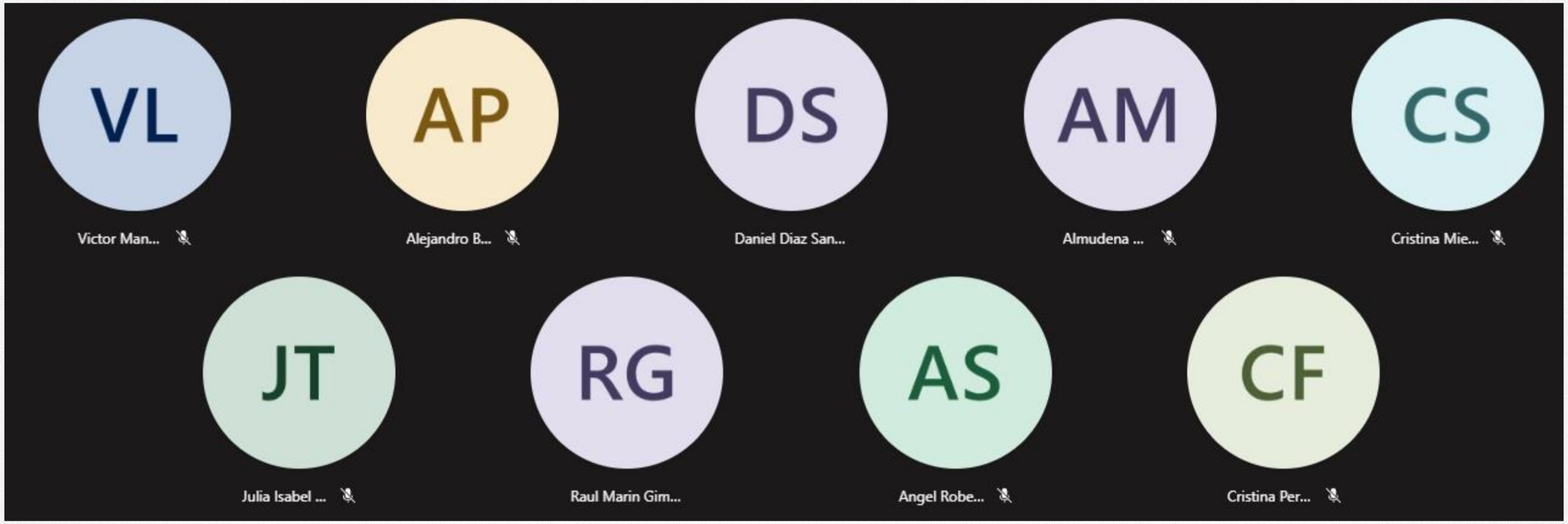


mylist = ['abcd', 786 , 2.23, 'john', 70.2]

mylist.append(10)
mylist.pop(0)

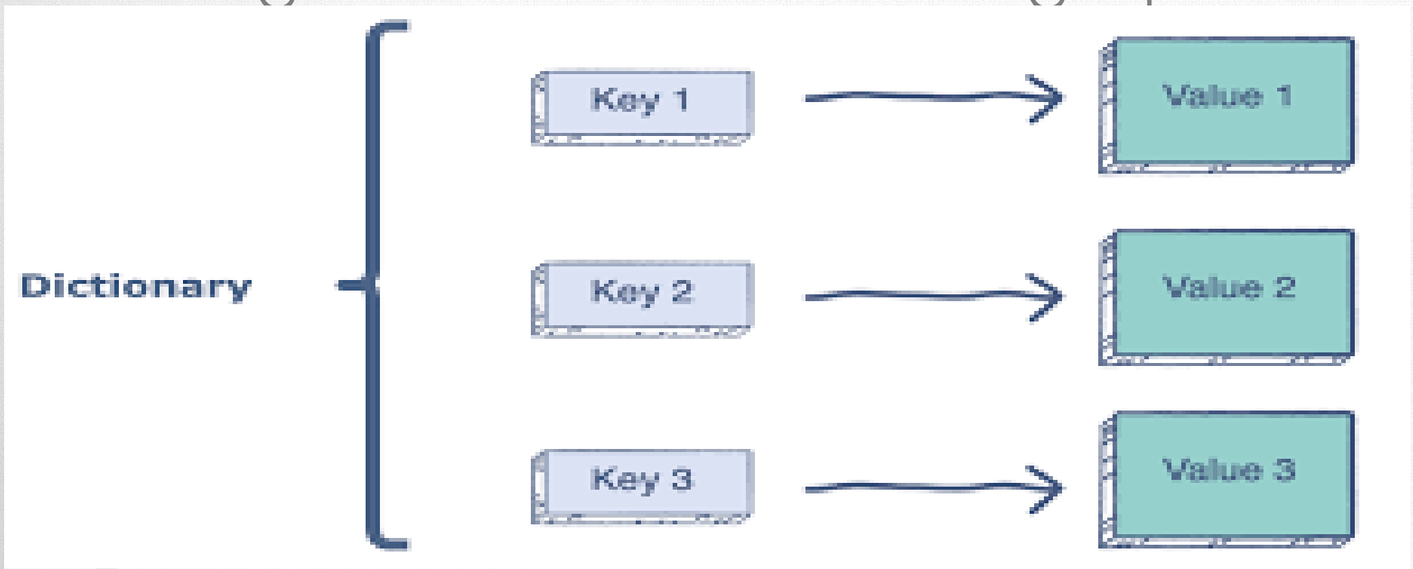

Data Structures in Python
Built-in Data Structures
User-Defined Data Structures
List
Dictionary
Tuple
Set
Stack
Queue
Tree
Linked List
Graph
HashMap

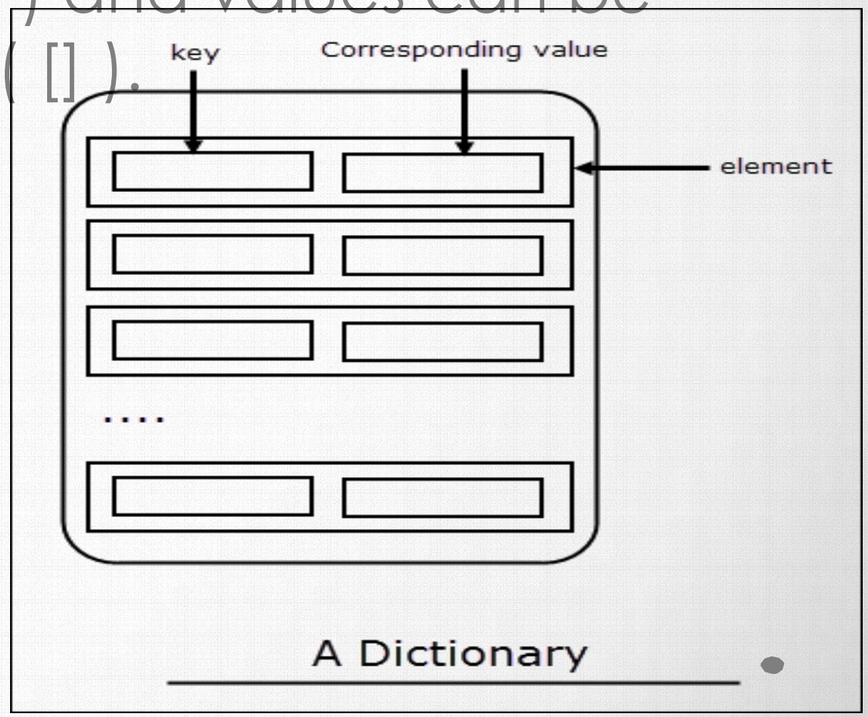
SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY


VL
Victor Man...
AP
Alejandro B...
DS
Daniel Diaz San...
AM
Almudena ...
CS
Cristina Mie...
JT
Julia Isabel ...
RG
Raul Marin Gim...
AS
Angel Robe...
CF
Cristina Per...

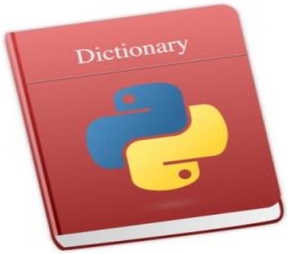
Python Dictionary:

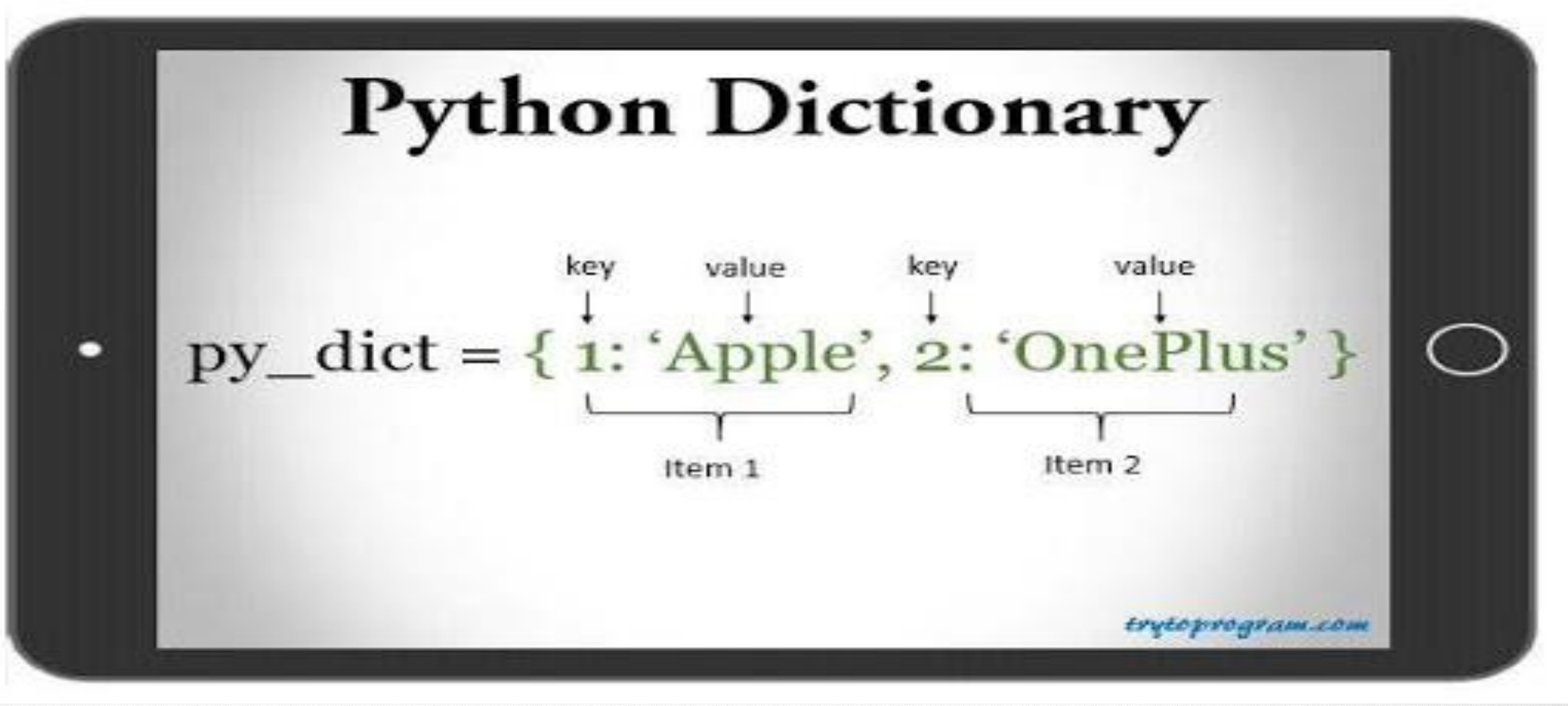
Python 's dictionaries are hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs.
Keys can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).



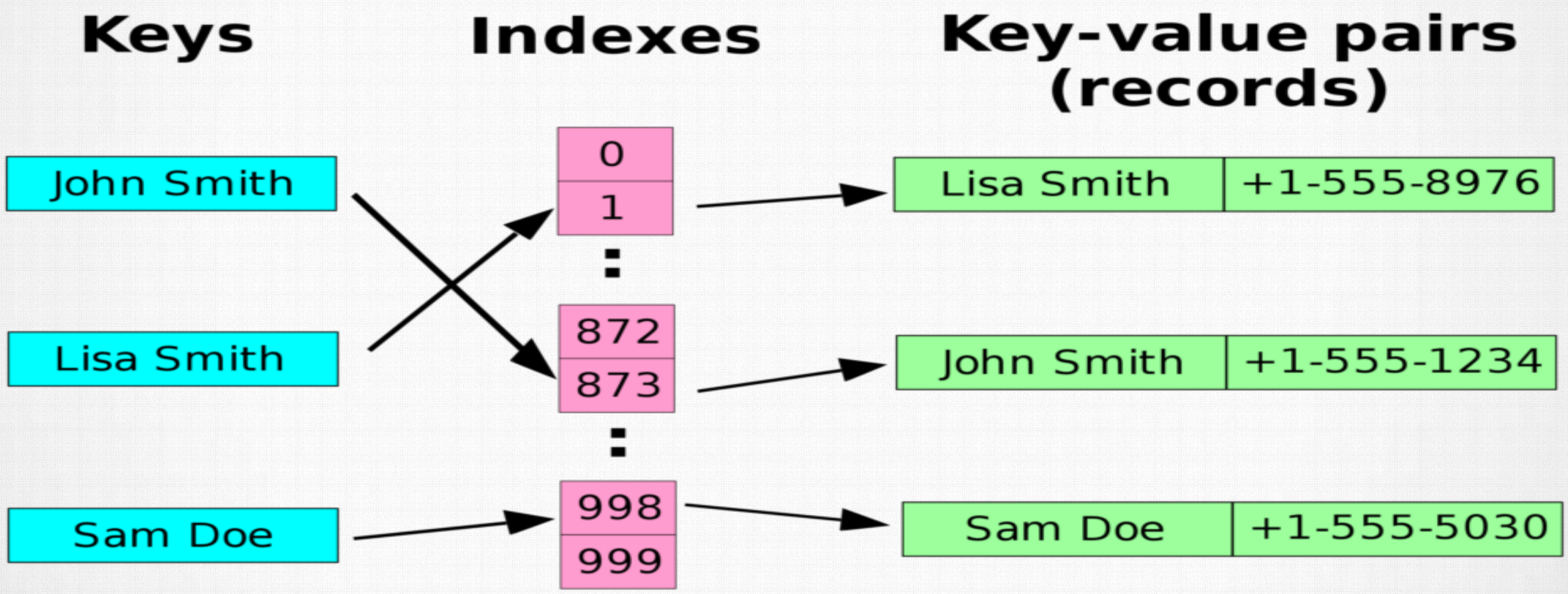


Python Dictionary - Syntax





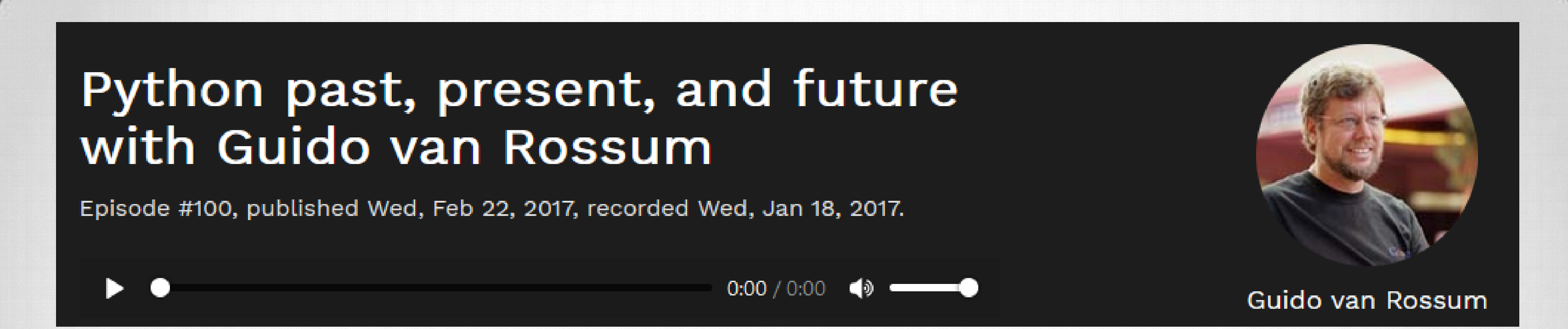
Python Dictionary – Is it efficient?



The fastest structure to find an element by key!

Python past, present, and future with Guido van Rossum

Episode #100, published Wed, Feb 22, 2017, recorded Wed, Jan 18, 2017.



Guido van Rossum

Which of the following statements are true?

a) Dictionaries do not have much value in Python

b) Dictionaries are a fundamental part of Python

c) Dictionaries are used extensively in applications

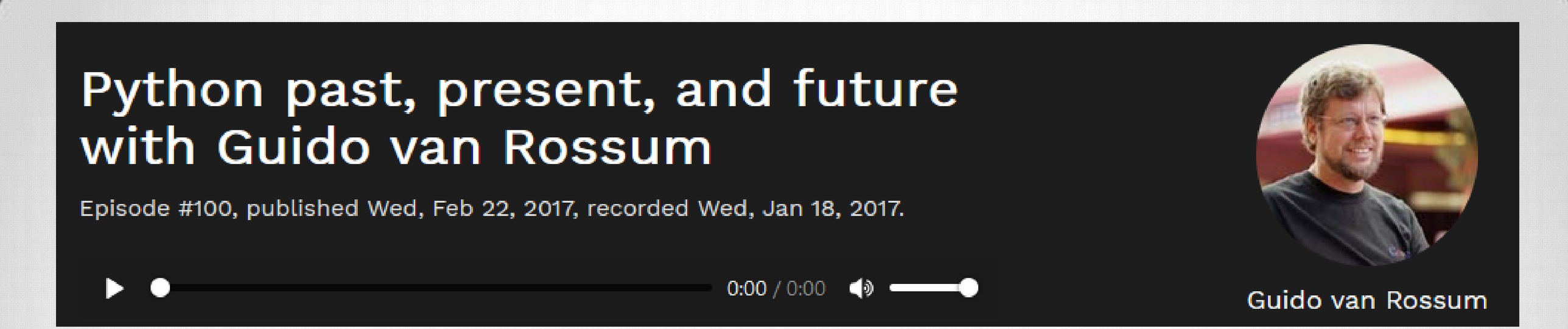
d) Dictionaries are not at par with the tech innovation

Correct answer at 49:10

https://talkpython.fm/episodes/show/100/python-past-present-and-future-with-guido-van-rossum

Python past, present, and future with Guido van Rossum

Episode #100, published Wed, Feb 22, 2017, recorded Wed, Jan 18, 2017.



Which of the following statements are true?

a) Dictionaries do not have much value in Python

b) Dictionaries are a fundamental part of Python

c) Dictionaries are used extensively in applications

d) Dictionaries are not at par with the tech innovation

Correct answer: b and c at 49:10

https://talkpython.fm/episodes/show/100/python-past-present-and-future-with-guido-van-rossum

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

By Prof. Manoel Gadi

50%	30%	20%
Good / Appropriate	Good / Appropriate	Faster / +Memory
Algorithm	**Data Structure**	**Computer**

Like Constants!

Pure Python (comes within the language)

	list	tuple	dictionary	set
COMMON:		Collection of 0, 1 or many of any combination of types of Python scalars		
DIFFERENTIAL:	Mutable, Ordered [0,n-1]	Immutable, Ordered [0,n-1]	Key is Unique and Unordered (key immutable / Value= Mutable)	Unique, Unordered
CREATE EMPTY:	list1=list(); list2=[]	tuple1=tuple(); tuple2=()	dict1=dict(); dict2={}	set1 = set()
CREATE:	list1 = [3,6,7,3,7]	tupple1 = (3,3,6,7,7)	dict1 ={3:'str',6:False}	set1 = {3,6,7}
INSERT:	list1.append(8)	-	dict1['Juan'] = 'Casado'	set1.add(8)
INSERT IN POSITION:	list1.insert(3,'value')	-	-	-
REMOVE:	list1.pop(1) / list1.remove(3)	-	del dict1['Juan']	set1.remove(8)

Python Ecosystem (libraries)

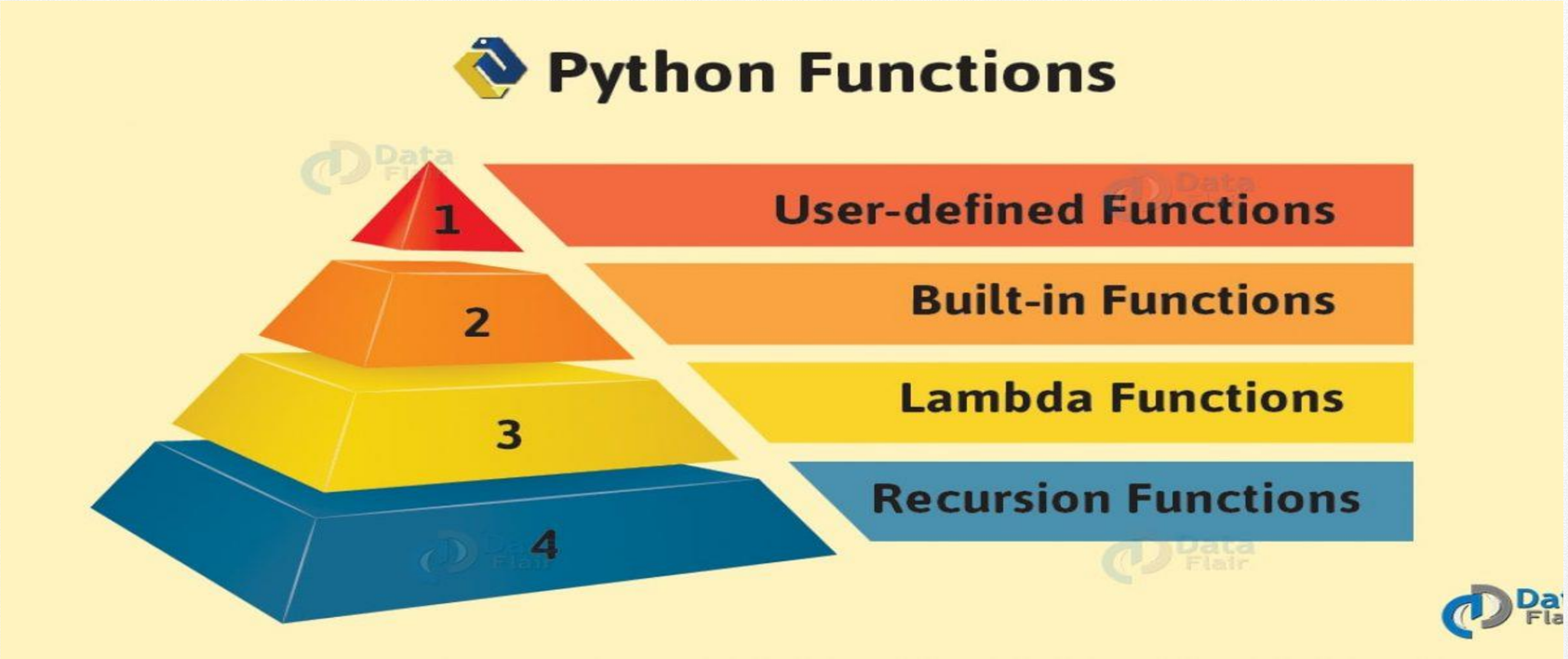
- Data frame (pandas)
- Key-Value (pickledb)
- HDFS (pytables)
- SQL

pwc

Universidad de Navarra

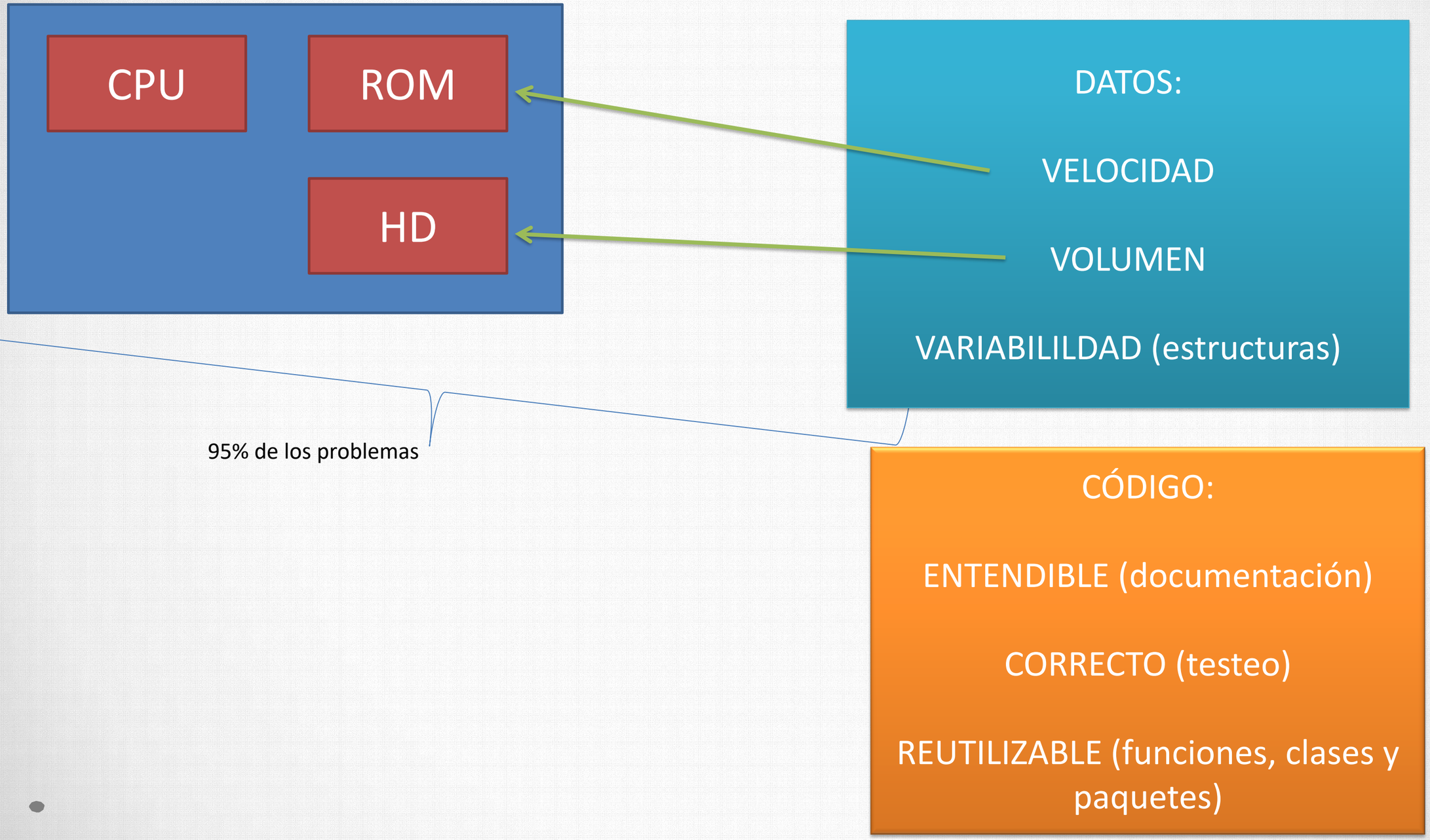
Functions

a named section of a program that performs a specific task



Functions:

A way of organizing code to make it more understandable, easy to use, reproducible and easier to share.


CPU
ROM
HD
DATOS:
VELOCIDAD
VOLUMEN
VARIABILILDAD (estructuras)
95% de los problemas
CÓDIGO:
ENTENDIBLE (documentación)
CORRECTO (testeo)
REUTILIZABLE (funciones, clases y paquetes)

list: delete items from the end of a list called myList: myList.pop()

dictionary: delete a key-value pair from the high_lows dictionary:
del high_lows[key]

set: delete a item from a set: myset.remove(item)

Example:

```
# language set
language = {'English', 'French', 'German'}
        # removing 'German' from language
language.remove('German')
        # Updated language set
print('Updated language set:', language)
```

About del high_lows[key]

```
del something # (1) It is procedural programming (PP)
        del(something) # (2) It is functional programming (FP)
        something.del() # (3) It is object oriented programming (OOP)
```

3 different paradigms, and Python is 80% (3 - OOP), 19% (2 - FP) and some bizarre cases it is (1 - PP) like in del high_lows[key]!!!

Normally (3) means that the function is specific to the object therefore called **method**, (2) means that a function can be applied to more than one type of object and (1) it means that Python has to call low level operations to do the trick.

functions vs. methods

A function looks like this: function(something)
And a method looks like this: something.method()

```
>>> a = ' BiG MuG '
>>> a.lower()
' big mug '
>>> a.upper()
' BIG MUG '
>>> a.strip()
'BiG MuG'
>>> a.strip().upper()
'BIG MUG'
>>> a.strip().upper().split(' ')
['BIG', 'MUG']
>>> ';'.join(a.strip().upper().split(' '))
'BIG;MUG'
```

A function can be applied to one or more type of things (objects), while methods are specific and can only be applied to "things" (objects) of one given type (one given class)

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

User defined functions


```
1   2              3    4
def function_name(args):
    #code in function
          5
```

```
(bottom)
1  def shake():
2      print('bugaloo')
3
4  shake()
5
```

Function & body

Function call

Debug I/O | Exceptions | Python Shell | Messages | OS Commands

Debug I/O (stdin, stdout, stderr) appears below

```
bugaloo
```

Function output

Line 5 Col 0 - [User]

SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

```
#### FUNCTIONS AND NOTHING - None is a type in python
#### FUNCIONES Y LA NADA - None es un tipo en python
def func1():
    """ this is my own doc string - it becomes available to anyone who calls help(func1). """
    pass

if func1()==None:
    print("It returned the 'nothing'! / Me ha devuelto la 'nada'!")
else:
    print("It returned something!")

help(func1)
```

FUNCTIONS AND NOTHING - None is a type in python
FUNCIONES Y LA NADA - None es un tipo en python
def func1():
""" This is my own doc string - it becomes available to anyone who calls help(func1)."""
pass

if func1()==None:
print("It returned the 'nothing'! / Me ha devuelto la 'nada'!")
else:
print("It returned something!")

help(func1)

Numeric
Strings
Constructors
Iterables (Lists, Dicts,…)

Built-in Functions

Python Functioning
Iinput / Output

abs() – Numeric
all()
any()
ascii()
bin()
bool() - Constructors
breakpoint()
bytearray()
bytes() - Constructors
callable()
chr() - Strings
classmethod()
compile() - Python Functioning
complex() - Constructors
delattr()
dict() - Constructors
dir()
divmod()
enumerate()
eval() - Python Functioning
exec()
filter()
float()
format() - Strings
frozenset()
getattr()
globals()
hasattr()
hash()
help()
hex() - Constructors
id()
input() - Iinput / Output
int()
isinstance()
issubclass()
iter()
len()
list() - Constructors
locals()
map()
max() -Numeric
memoryview()
min() - Numeric
next()
object()
oct()
open() - Iinput / Output
ord()
pow()
print() - Iinput / Output
property()
range() - Iterables
repr()
reversed() - Strings
round()
set() - Constructors
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()
tuple()
type()
vars()
zip()
__import__()

https://docs.python.org/3/library/functions.html

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

```
#### FUNCTIONS - DEFAULT PARAMETERS & RECURSION- THE ZEN OF PYTHON.
#### FUNCIONES - PARAMETROS DEFAULT & RECURSIÓN - EL ZEN DE PYTHON.
def func2(y=True,z_lista=[], x=10, recursion_level=1):
    if (recursion_level<2):
        func2(x,y,z_lista.copy(),recursion_level+1)
    d_lista[2] = recursion_level
    return(x,y,z_lista)

a,b,c=func2()
print(a,b,c)

a,b,c=func2(True,d_lista.copy(),12,recursion_level=0)
print(a,b,c)
```

FUNCTIONS - DEFAULT PARAMETERS & RECURSION- THE ZEN OF PYTHON.
FUNCIONES - PARAMETROS DEFAULT & RECURSIÓN - EL ZEN DE PYTHON.
def func2(y=True,z_lista=[], x=10, recursion_level=1):
    if (recursion_level<2):
        func2(x,y,z_lista.copy(),recursion_level+1)
    d_lista[2] = recursion_level
    return(x,y,z_lista)

a,b,c=func2()
print(a,b,c)

a,b,c=func2(True,d_lista.copy(),12,recursion_level=0)
print(a,b,c)

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

```
#### TUPLES & FUNCTIONS - PARAMETERS and RETURN VIA TUPLES
#### FUNCIONES - PARAMETROS y RETURN VIA TUPLAS.
d_list = [1,"a",True]

# On the right side you are packaging in a tuple.
# / En la parte derecha está empaquetando en una tupla.
# On the left side you are unpacking the tuple into variables.
# / En la parte izquierda está desempaquetando de la tupla en variables.
(d_list[2], d_list[1], d_list[0]) = (d_list[0], d_list[1], d_list[2])

d_tuple = (1,"a",True)
d_list[1] ="ready!"
print(d_list[1],d_tuple[1])

def func2(x,y,z):
    return(x,y,z)

a,b,c=func2(12,True,"Pepe")
print(a,b,c)
```


```
d_list = [1,"a",True]
(d_list[2], d_list[1], d_list[0]) = (d_list[0], d_list[1], d_list[2])

d_tuple = (1,"a",True)
d_list[1] ="ready!"
print(d_list[1],d_tuple[1])

def func2(x,y,z):
    return(x,y,z)

a,b,c=func2(12,True,"Pepe")
print(a,b,c)
```

MATERIAL EXTRA

https://www.youtube.com/watch?v=hgI0p1zf31k