- Guía Básica para programación -

Con Python

Una amigable introducción a la programación por Pythonízate.com

Revisión 202107, por Alí Adame

Guía Básica para Programación (con Python)
por Alí Adame Cantorán – ali@pythonizate.com

Índice de contenido

Antes de empezar (una bienvenida)

El curso de programación básica de Pythonizate.com es un esfuerzo por voluntarios sin fines de lucro que va a existir siempre que el mundo necesite más programadores de los que hay en la actualidad.

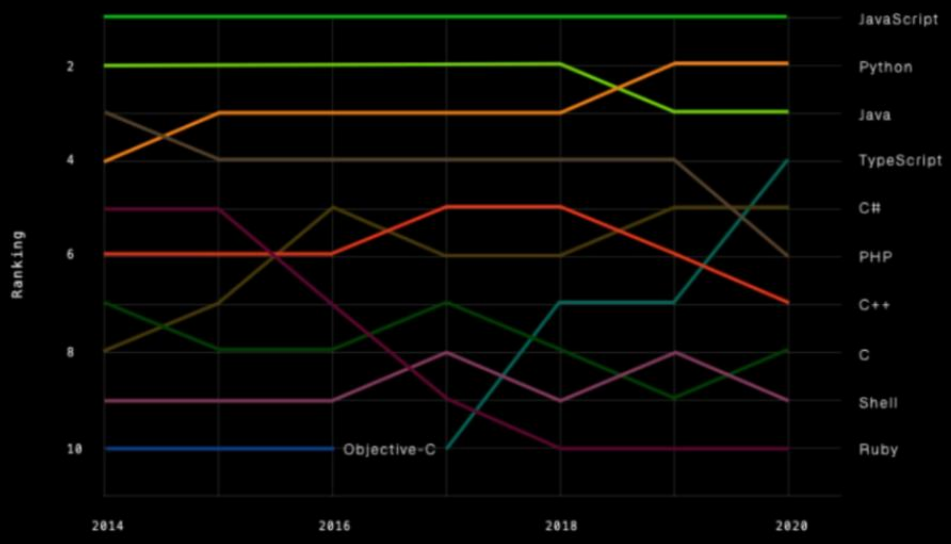
El material de este manual del curso contiene retos a completar a manera de ejercicios. Todos o la gran mayoría presentan múltiples soluciones, pues para programar, incluso siendo eficiente en código, se puede proceder de diversas formas. Encontrarás información sobre la solución en cada ejercicio.

Es importante que te tomes el tiempo antes de iniciar cualquier curso o manual para preparar tu ambiente de trabajo. En caso de poder usar equipo de cómputo toma tu tiempo para instalar Python **(o si cuentas con internet permanentemente puedes usar el sitio Trinket.io que es un buen sustituto inicial para ejecutar Python desde la web que no requiere instalación)**

Otro particular asunto que se recomienda para seguir este manual atentamente es alejarse de distractores y de preferencia tener un espacio iluminado y bien ventilado donde consultar el material donde se pueda leer cómodamente. El área de estudio juega un papel fundamental en el aprendizaje.

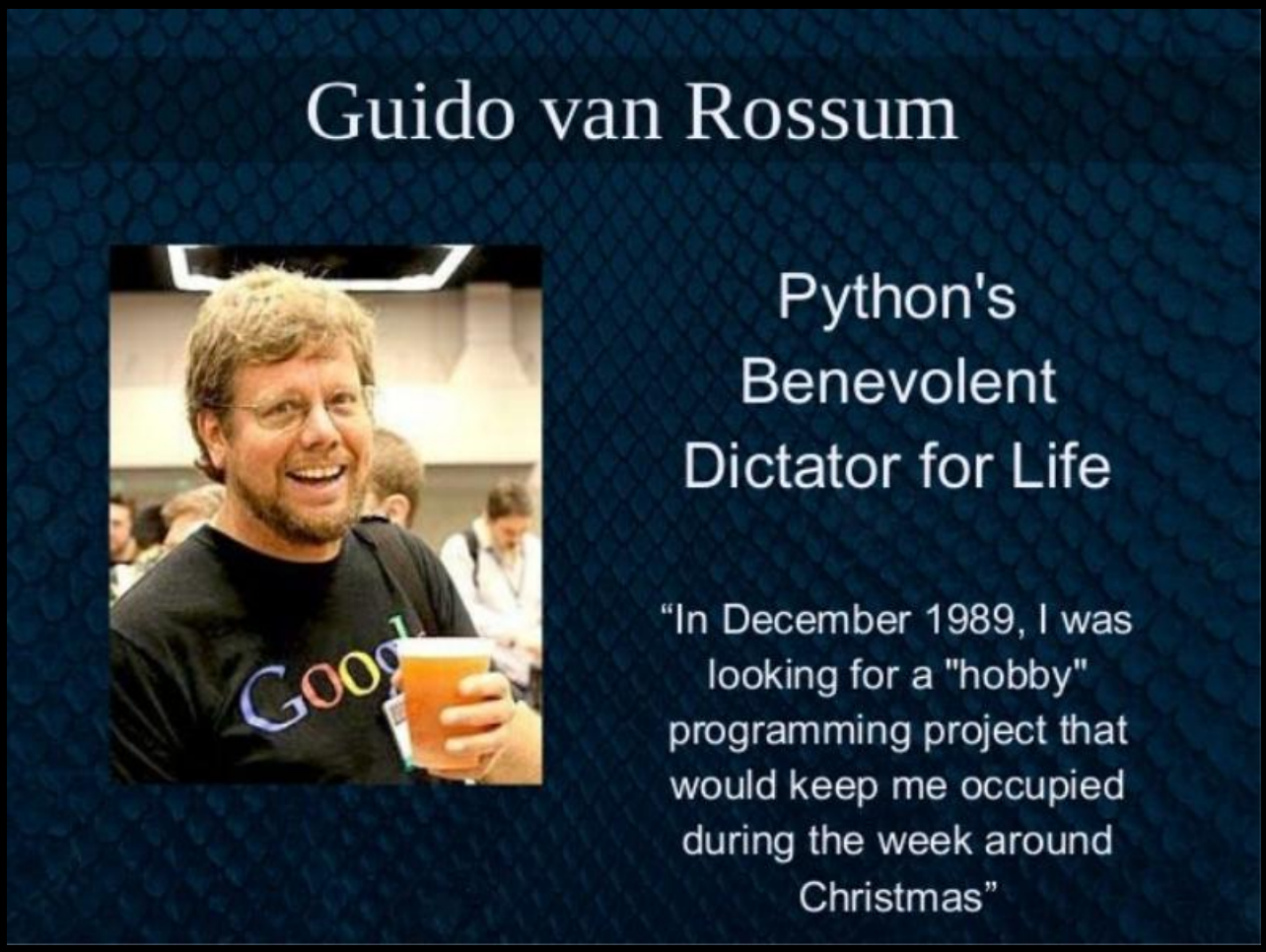
Por último te pedimos te diviertas y si no entiendes una parte vuelve a leer, si sigue sin quedar claro toma una pausa y luego vuelve a él. Si después de la pausa sigue sin tener sentido deja una nota para volver más adelante y continúa, con suerte y más adelante conforme veas más contenido similar tal vez lo asimiles mediante otro planteamiento. No pierdas el entusiasmo y mantente curioso.

Roma no se hizo en un día; sé paciente con tus logros y constante con tus intentos de aprendizaje. **No olvides que sea cual sea el motivo por el cual quieres aprender a programar al inicio no será fácil, pero con el tiempo lo será.**
Te dejamos esta gráfica mostrando que al escoger Python no te equivocaste.



Lenguajes de programación por popularidad disponibles en GitHub

Antes de comenzar te queremos presentar a Guido Van Rossum, creador del lenguaje, y autonombrado "Dictador Benevolente de por Vida de Python", en 1989 se encargó de mantenerse ocupado en sus vacaciones de Navidad y al día de hoy su ocupación nos mantiene ocupados también a nosotros.



Por último te invitamos a que nos contactes y si gustas colabores con nosotros, nuestro sitio es llevado a ti por voluntarios que entienden que en la programación hay cabida para todos, sin importar raza, género, religión o condición social, porque en realidad, desde la invención del cómputo y de la web existe una deuda técnica de especialistas que no acaba por ser satisfecha sin importar cuántos nuevos graduados y programadores sin título haya, al menos en el ritmo actual.

Gracias por leernos y esperamos saber de ti.
Que disfrutes el manual y que programes y aprendas mucho.

Instalación de Python para los ejercicios de programación

Antes de empezar a programar necesitaremos instalar Python en el ordenador. Como no sé qué sistema operativo está usando el lector, explicaré cómo proceder a la instalación en cada caso.

Instalación en Linux

En Linux la instalación resulta realmente sencilla. Si eres usuario de Ubuntu (Debian y derivados), basta con abrir la terminal y escribir lo siguiente:

sudo apt-get install python3

Si eres usuario de otra distribución no puedo guiarte desde aquí, ya que puede variar el gestor de paquetes y el nombre de la paquetería. Escríbeme a mi correo si necesitas ayuda y estaré encantado de echarte un cable.

Instalación en Mac OS

Python viene preinstalado en este sistema. Si tu versión estuviera desactualizada deberás descargar uno de los siguientes paquetes según te convenga:

Si tienes un Macintosh con procesador Intel debes descargar el siguiente instalador:

http://www.python.org/ftp/python/3.2.3/python-3.2.3-macosx10.6.dmg

Si por el contrario tu máquina tiene un procesador PowerPC, debes descargar este:

http://www.python.org/ftp/python/3.2.3/python-3.2.3-macosx10.3.dmg

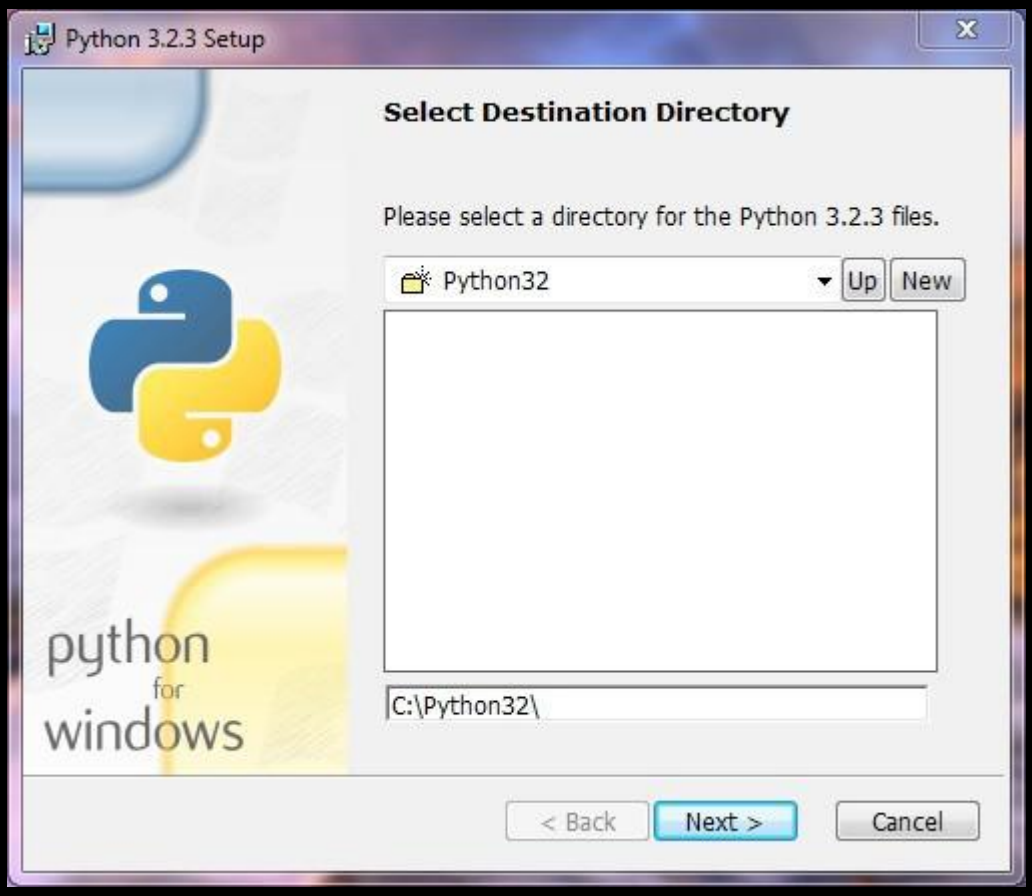
[Para ver la guía de instalación en Windows pasar a la siguiente página]

Instalación en Windows

Por suerte (o por desgracia) todo el mundo dispone de una copia de Windows en casa, incluso yo; así que en esta instalación si os puedo guiar.

Python no viene preinstalado en Windows, por lo que obligatoriamente deberás descargar este paquete:

http://www.python.org/ftp/python/3.2.3/python-3.2.3.msi



La instalación en Windows no tiene mayor complicación: basta con hacer clic en "Siguiente" repetidas veces hasta finalizar.

Y con esto ya tenemos instalado Python en nuestro ordenador.

¿Cómo hago mi programa?

Un programa está compuesto por códigos, órdenes simples que las personas podemos comprender. Por ejemplo:

- Lenguaje humano:

Orden: Dime, ¿cuánto son 2+2?
Respuesta: Son 4.

- Lenguaje Python:

Orden: print(2+2)
Respuesta: 4

Fácil, ¿verdad?.

Para escribir dichos códigos nosotros usaremos un editor de texto. ¿Conoces el *bloc de notas* de Windows? pues ese sería un ejemplo.

Pero el *bloc de notas* es muy malo, así que nosotros vamos a usar otro editor.

Editores para Linux

En Linux tenemos magníficos editores como *Gedit* (GNOME) y *Kate* (KDE).
Si eres un enamorado de la terminal también puedes usar *Pico*.

Asumo que los usuarios de Linux saben encontrar estos programas en su sistema o, en caso de no tenerlos, saben instalarlos.

Editores para MacOS

Los usuarios de Mac también pueden disfrutar de *Gedit*:

- Para Tiger (10.4, Intel): descargar
- Para Leopard (10.5): descargar
- Para Snow Leopard (10.6) y posterior: descargar

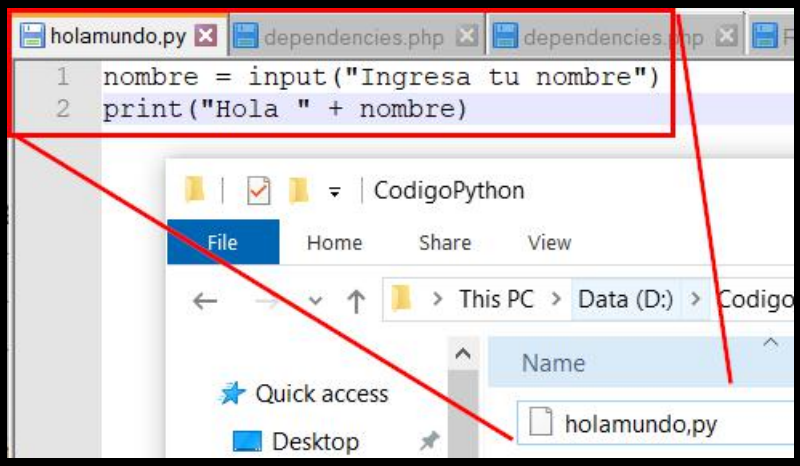
Editores para Windows

Para Windows hay gran variedad, aunque yo me decanto por estos dos:

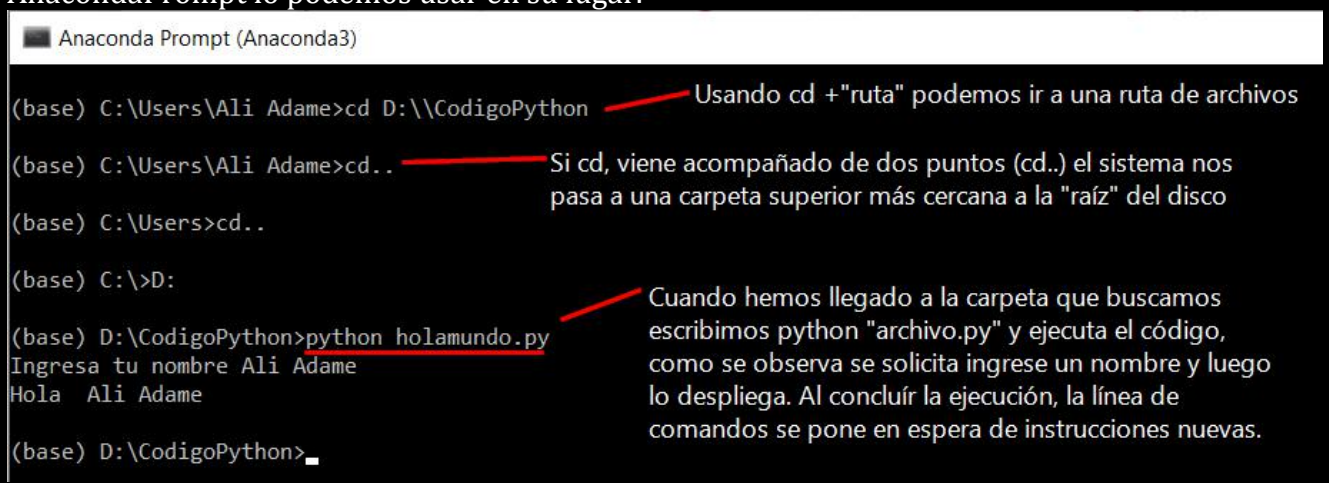
- Gedit: descargar
- Notepad++: descargar

Si ya tienes tu editor listo, vamos a ver rápidamente cómo ejecutar tu código Python (en caso que no vayas a usar Trinket.io deberás tener instalado el lenguaje Python en tu sistema operativo, como se detalló previamente).

Para ello se ha creado un archivo en nuestro sistema de archivos con ayuda de Notepad++, el archivo creado se llama "holamundo.py", se creó en nuestro disco "D" del equipo en la carpeta llamada "CodigoPython", el contenido completo del archivo se resume a dos lineas, numeradas en Notepad++ del 1 al 2, de manera que la ruta completa al archivo es **D:\\CodigoPython*holamundo.py***

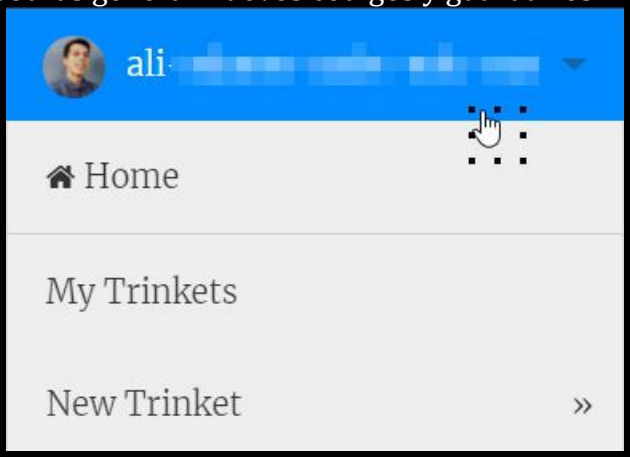


Cuando queremos ejecutar un programa solo debemos entrar usando una herramienta de comandos y usar el comando "python (archivo a ejecutar, como parametro)" y eso ejecutará nuestro código, en Windows eso lo podemos hacer con la herramienta CMD (escribir CMD en la busqueda de Windows), PowerShell o, si tenemos otro software que soporte comandos, como AnacondaPrompt lo podemos usar en su lugar.



Usando el editor en línea de Python disponible en Trinket.io

Para sacar mayor provecho a la experiencia usando Trinket (https://trinket.io), se recomienda usarlo con una cuenta de Google o crear una cuenta en el sitio, de esa forma se hace posible guardar nuestro código. Al ingresar usando la opción de menú de la esquina superior derecha podrás generar nuevos códigos y guardarlos.



El siguiente es el editor de código disponible para diseñar y ejecutar el código en Trinket.io, aparece cuando presionamos la opción "New Trinket".

Haciendo lo anterior ya tendremos lo necesario para empezar a aprender sea con herramientas en línea, fuera de línea o si escogemos escribir el código a mano ya habrá una noción sobre cómo hacerlo.

Introducción a la resolución de problemas: los algoritmos

Casi todos los problemas tienen solución, de una u otra forma los humanos con nuestro ingenio hemos probado muchas maneras de resolverlos.

Cuando pensamos en un problema como un proceso identificamos tres elementos: entradas, procesamientos y salidas.

Si pensamos en el asunto con una analogía veríamos que por ejemplo para hacer el proceso de hacer una pizza tenemos tres elementos: las entradas (los ingredientes que usaremos para nuestra creación, los procesos (los pasos que ordenados dan lugar a una creación, en este caso para hornear y armar la pizza) y la salida (el producto terminado para nuestro consumo).

Cuando queremos hacer una pizza vegetariana, seguimos pasos distintos a una pizza de pepperonni, una ligera variación en los pasos y obtenemos un producto distinto. El proceso para hacer algo en nuestra computadora, así como la receta para hacer un tipo de pizza lo conocemos como algoritmo.

Todos los algoritmos tienen entradas, realizan procesamientos y conducen a una salida (o varias). Las fotos que subes a tu red social favorita (entrada), se procesan con filtros o cambios de tamaño (procesamiento) y terminan logrando ser vistas por tus amigos (salida). Todo es un algoritmo para tu computadora y para internet. Los algoritmos una vez creados, se prueban y se despliegan cuando están listos para ser usados.

Fundamentos: Propiedades de los algoritmos

Las propiedades de un algoritmo son las siguientes:

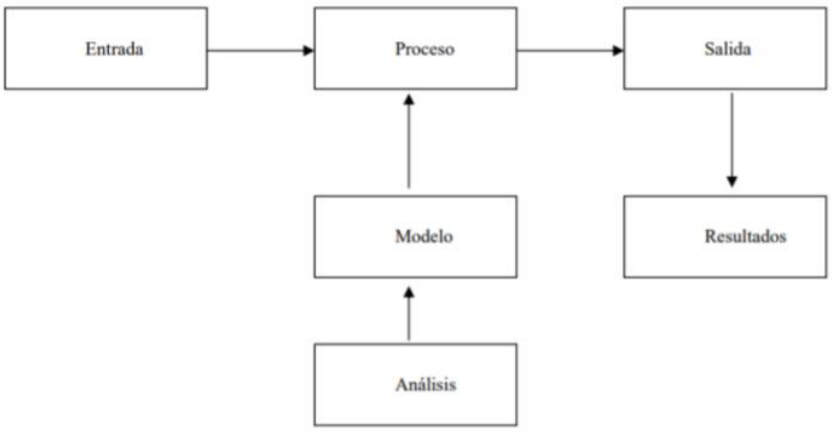
- El algoritmo debe ser preciso e indicar el orden de realización de cada paso
- El algoritmo debe ser definido, si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- El algoritmo debe ser finito, si se sigue un algoritmo se debe terminar en algún momento; o sea debe tener un número finito de pasos.

PYTHONIZATE
Aprende a programar sin perder la cabeza

Los algoritmos generan información o transforman datos. Cumplen una función establecida.

Fundamentos: Propiedades de los algoritmos

Un algoritmo debe estar planeado como un sistema de información.



Representación de un algoritmo como un sistema de Información

Fundamentos: Ejemplo de un algoritmo

A continuación se presenta ejemplo de un algoritmo representado mediante el uso de pseudocódigo.

```
algoritmo Sumar

  variables
  entero a, b, c

inicio
  escribir( "Introduzca el primer número (entero): " )
  leer( a )
  escribir( "Introduzca el segundo número (entero): " )
  leer( b )
  c ← a + b
  escribir( "La suma es: ", c )
fin
```

El pseudocódigo es una manera de representar algoritmos, son un precursor de los diagramas de flujo que usaremos en este manual, pues también son fáciles de entender.

Fundamentos: Identificadores

La mayoría de los elementos de un algoritmo escrito en pseudocódigo se diferencian entre sí por su nombre. Por ejemplo, los tipos de datos básicos se nombran como:

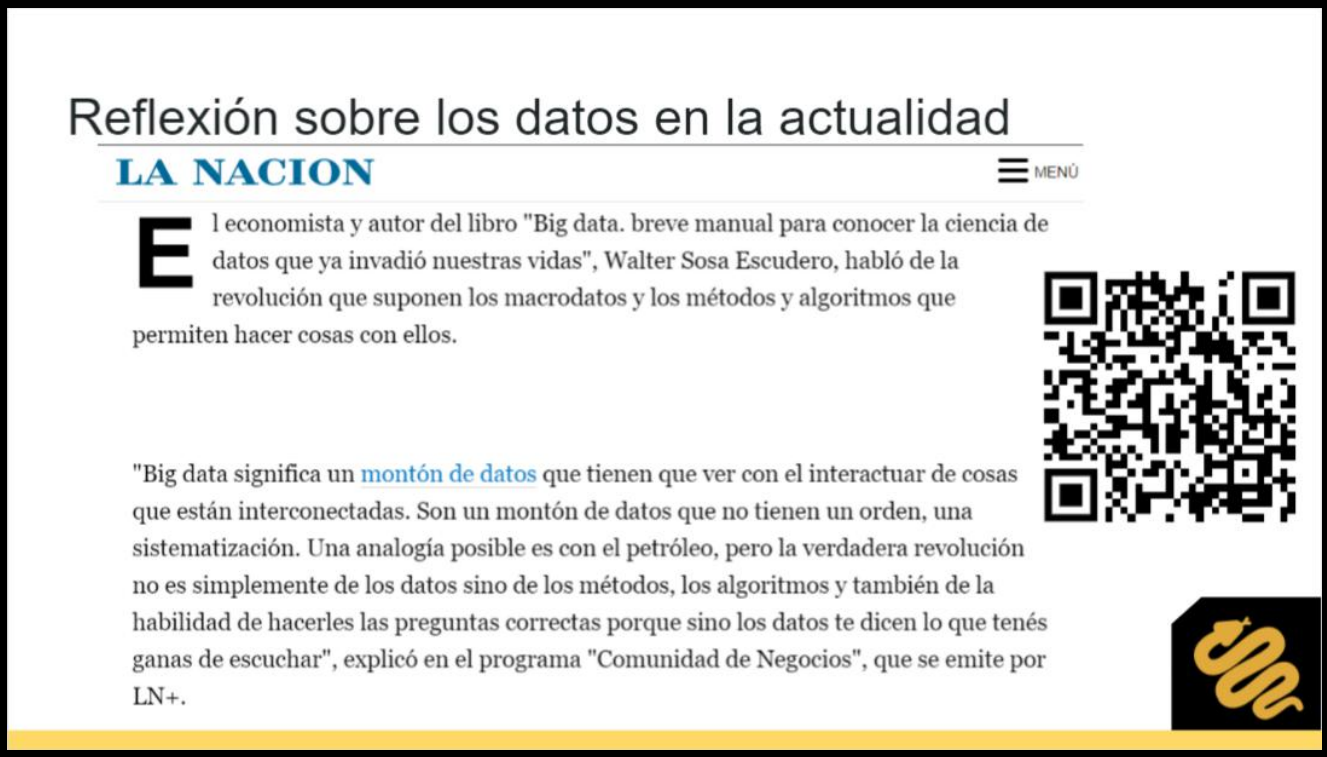
- Entero
- Real
- Logico
- Caracter

Un identificador es el nombre que se le da a un elemento de un algoritmo (o programa)
entero, real, lógico y caracter son identificadores predefinidos, forman parte del lenguaje algorítmico

Lo tipos mencionados anteriormente son ejemplos de identificadores

Datos, tipos de datos y variables

Nuestros algoritmos se alimentan de datos, no los vemos, no los podemos tocar pero están ahí, ¿qué son los datos? Algunos los llaman “el petróleo del siglo XXI”



Los datos pueden ser simples o compuestos y se descomponen en tipos a partir de su naturaleza

Tipos de datos simples

Tipos de datos simples (sin *estructura*) **en pseudocódigo:**

Predefinidos (*estándares*):
- *Numéricos:*
  - *Entero* (`entero`)
  - *Real* (`real`)
- *Lógico* (`logico`)
- *Carácter* (`caracter`)

Definidos por el programador (no *estándares*):
- *Subrangos* (`subrango`)
- *Enumerados* (`enumerado`)

© carlospes.com

En informática tenemos dos tipos de datos simples:

- Datos predefinidos
- Datos definidos por el programador

Se acompañan ejemplos de tipos de datos, en los ejemplos usando otro lenguaje diferente a Python, más usado para la web: Javascript, verás muy pronto que con Python el código es igual de simple

Tipos de datos: entero

Un dato de tipo entero es aquel que puede tomar por valor un número perteneciente al conjunto de los números enteros (Z), el cual está formado por los números naturales, su opuestos (números negativos) y el cero.

JS

```
var entero = 9;

// forzar entero
entero = parseInt(entero)
alert(entero);
```

Z = { ..., -3, -2, -1, 0, 1, 2, 3, ... }

PYTHONIZATE
Aprende a programar sin perder la cabeza

Tipos de datos: real

Un dato de tipo real es aquel que puede tomar por valor un número perteneciente al conjunto de los números reales (R), el cual está formado por los números racionales e irracionales.

Racionales: 9/2

Irracionales: Pi

JS

```
var entero1 = 9;
var entero2 = 2;

// forzar real
var real =
    parseFloat(entero1/entero2);
alert(real);
```

En términos de memoria de nuestra computadora o celular, almacenar un valor entero es más sencillo que un dato de tipo real, ¿por qué será?

Tipos de datos: lógico

En programación, un dato de tipo lógico es aquel que puede tomar por valor sólo uno de los dos siguientes:

{ verdadero, falso }.

Es popular para indicar estados binarios

JS

```
var verdadero = 1;

// forzar binario en JS
var logico =
    (verdadero == 1);
alert(logico);
```

PYTHONIZATE
Aprende a programar sin perder la cabeza

Un dato lógico solo tiene dos posibles valores: o es verdadero o es falso

Tipos de datos: caracter

Un dato de tipo carácter es aquel que puede tomar por valor un caracter perteneciente al conjunto de los caracteres que puede representar el ordenador.

Los caracteres más populares son [A-Z0-9] *(lo anterior fue una "expresión regular", hace referencia a los caracteres de la A a la Z y del 0 al 9)*

```JS
var caracter = 'f';

alert(caracter);
```

Tipos de datos: cadena

Un dato de tipo cadena es aquel que pueden tomar por valor una secuencia de caracteres. En pseudocódigo, el valor de un dato de tipo cadena se puede representar entre comillas simples (') o dobles ("). Sin embargo, lenguajes como Java si hacen distinción entre uno y otro

```JS
var cadena = 'Fernando';

alert(cadena);
```

PYTHONIZATE
Aprende a programar sin perder la cabeza

Un dato de tipo cadena, está **compuesto** por datos de tipo caracter.

Actividad no. 1
Nombre: _____

Instrucciones: Partiendo de lo revisado en el tema de tipos de datos indica si los siguientes datos son simples o compuestos
Ejemplo
Fecha, solo el año: **Simple**

1. - Fecha, con año, mes, día, horas y minutos: _____
2. - Tu fecha de nacimiento: _____
3. - Tu nombre:_____
4. - El nombre con apellidos de un amigo: _____
5. - El precio al que compras un celular, con centavos: _____
6. - Un número de identificación personal, como CURP: _____

Instrucciones: Indica los tipos de dato de los siguientes datos

Ejemplo
12.56: **Real**

1. - 123456: _____
2. - Me gustra estudiar: _____
3. - Verdadero: _____
4. - W : _____
5. - π : _____

Actividad no. 2
Nombre: _____

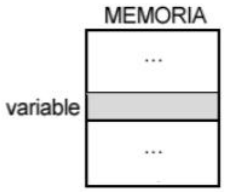
Instrucciones: Partiendo de lo observado en el tema de variables, indica "Verdadero" o "Falso" si el dato presentado en para cada ejercicio corresponde a una variable o no

Ejemplo
mesDelAnio = **Verdadero**

1. - nombre = _____
2. - fechaNacimiento = _____
3. - edad_actual = _____
4. - SIGNO_zodiacal = _____
5. - iva = _____
6. - altura = _____
7. - indice masa corporal = _____
8. - coeficiente Intelectual = _____
9. - GNI = _____
10. - CURP = _____

Variables, ¿qué son?

En programación, una variable representa a un espacio de memoria en el cual se puede almacenar un dato.



"Una variable es un elemento de datos con nombre cuyo valor puede cambiar durante el curso de la ejecución de un programa. Un nombre de variable debe seguir el convenio de denominación de un identificador (carácter alfabético o número y el signo de subrayado)"
- IBM

Las variables son importantes, pero tanto como almacenar los datos en variables es importante el nombre que se le dé a cada una porque facilita el trabajo en el futuro.

Variables, están en todo el código

El programador, cuando desarrolla un programa (o diseña un algoritmo), debe decidir:

- Los nombres de las variables que el programa necesita para realizar las tareas que se le han encomendado. Los nombres asignados deben ser descriptivos.
- El programador asigna tipos de datos que se almacenan a cada variable.

El valor que tome el dato almacenado en una variable puede cambiar tantas veces como sea necesario. Normalmente en otros lenguajes el tipo de dato de una variable no puede ser cambiado durante la ejecución de un programa. Cuando el lenguaje es de fuerte tipado.
No es el caso de Python, ahí todo se puede.

PYTHONIZATE
Aprende a programar sin perder la cabeza

Abordaremos el tema de las variables cuando comencemos con los diagramas de flujo, por lo que revisitar el material anterior es necesario.

Diagramas de flujo como herramienta para representar procesos

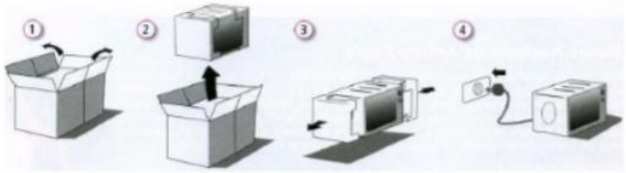
Representar los algoritmos siempre fue un desafío, no ahondaremos en los diversos antecedentes para ir directo al asunto, aunque es un tema fascinante y se presentó un ejemplo de un algoritmo en pseudocódigo.

Fundamentos: Diagramas

Textualmente, de acuerdo con la RAE un diagrama es:

"Representación gráfica de una sucesión de hecho y operaciones en un sistema".

Es decir que un diagrama, dependiendo su naturaleza contiene imágenes, dibujos, fotos o símbolos y en ocasiones también palabras.



Uso de los diagramas

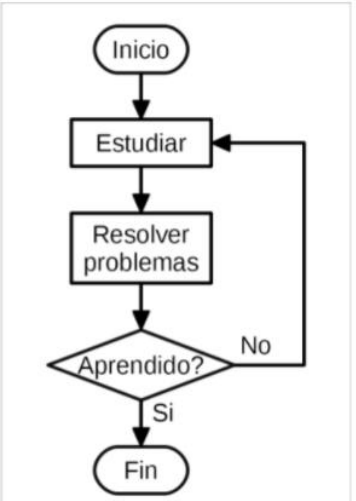
Podemos emplear un diagrama para:

- Representar los pasos de un proceso o procedimiento.
- Indicar la relación entre las operaciones y/o elementos.
- Representar un algoritmo.
- Entre otros...

PYTHONIZATE
Aprende a programar sin perder la cabeza

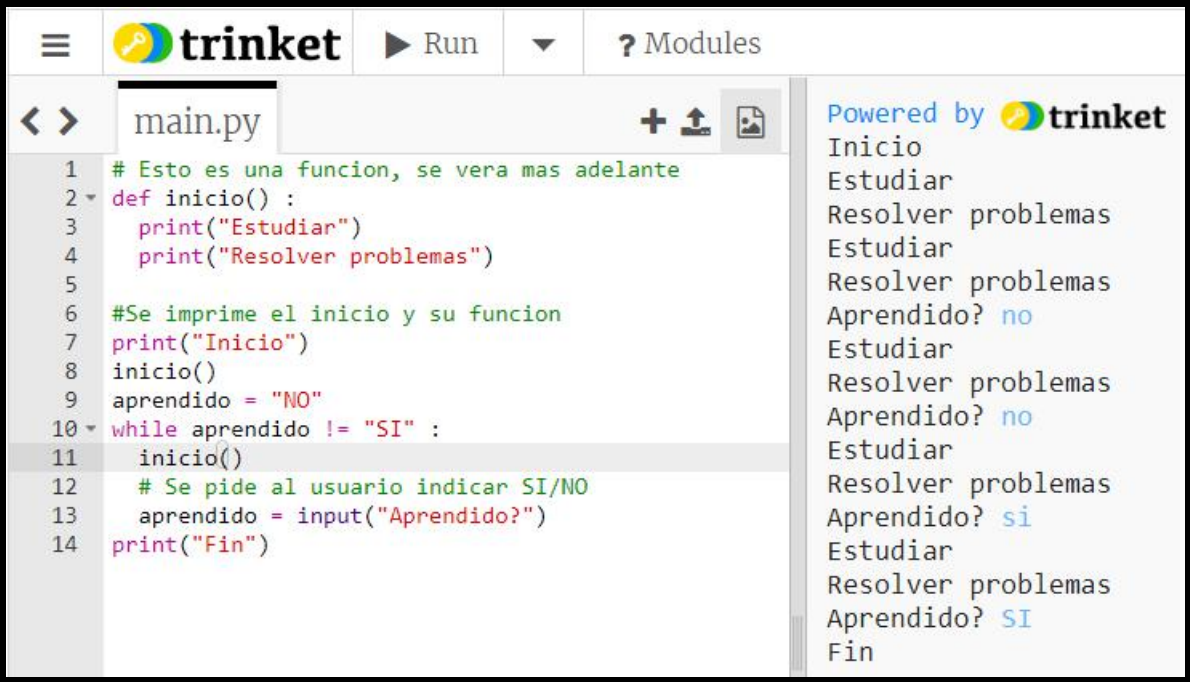
Diagramas de flujo

Una de las técnicas más utilizadas en el diseño de la solución de problemas es la de diagramas de flujo, los cuales nos ayudan de manera muy efectiva, ya que las soluciones posibles son observadas gráficamente; existen diferentes tipos de instrucciones que son posibles utilizar para la resolución de casos



PYTHONIZATE
Aprende a programar sin perder la cabeza

El diagrama anterior, cuando lo trasladamos a Trinket.io con programación, consta de una función y una estructura de repetición.



Simbología para diagramación

De manera metódica, se utilizan ciertos símbolos y elementos para expresar relaciones y comportamientos entre diversos elementos del diagrama como los siguientes:

- Asignaciones: Se utiliza una flecha dirigida

  Variable ← Expresión o valor asignado

  Las variables regularmente se inicializan con un valor por defecto, el valor por defecto implica valor antes de ejecutar nuestro diagrama

La asignación anterior en Python sería así:

```
variable = "Expresión o valor asignado"
```

Simbología para diagramación

- Asignaciones (continúa):

  En cada asignación se almacena el valor que corresponde a cada variable.

  Nombre ← "Camila"

  Nombre ← "Pepe"

PYTHONIZATE
Aprende a programar sin perder la cabeza

Pasado a Python el ejemplo anterior requeriremos escribir:

```
Nombre = "Camila"
Nombre = "Pepe"
```

Simbología para diagramación

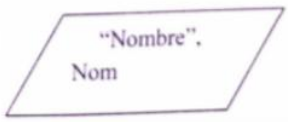
- Lectura de datos: Implica interpretar los datos a partir de una entrada, generalmente el teclado. Las lecturas se ingresan a memoria.

  Para la lectura de datos se utilizan paralelogramos

Símbolos de entrada de datos

Simbología para diagramación

- Ejemplo de entrada de datos con asignación de variable



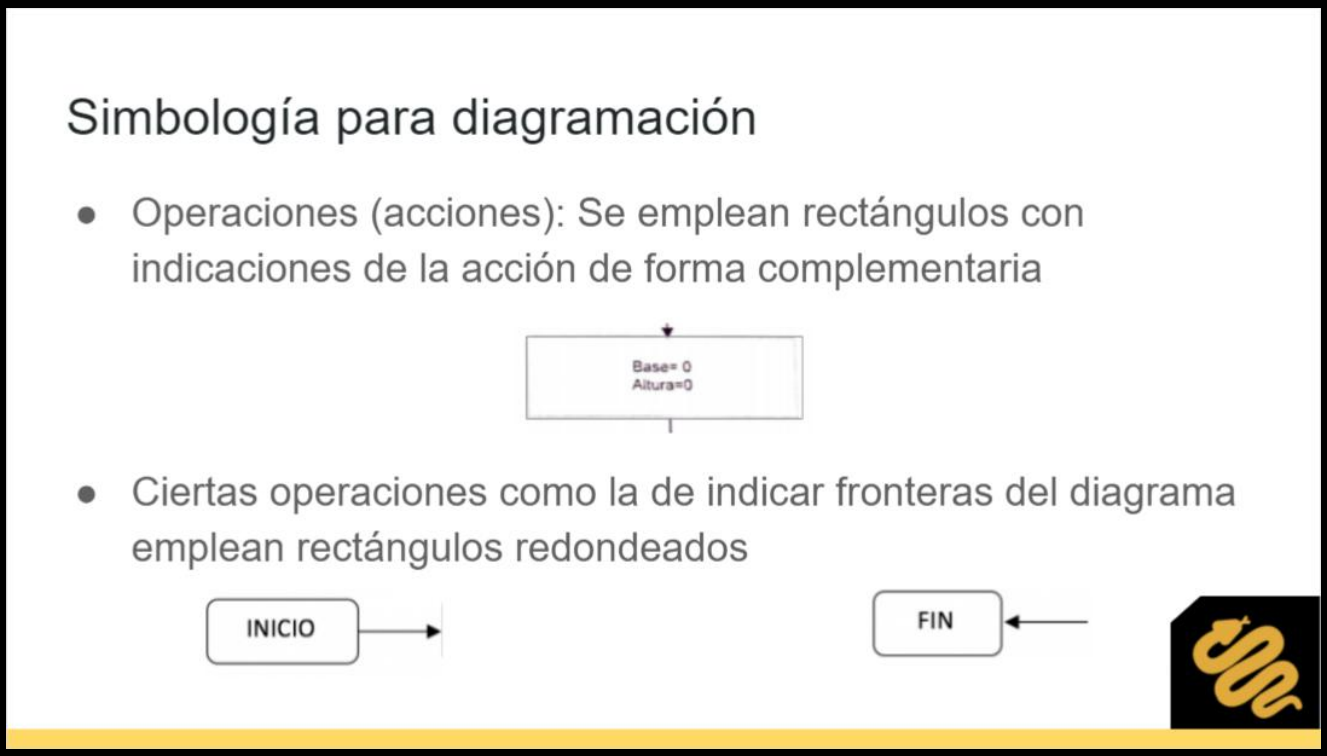
Ejemplo de símbolo de entrada de datos, con texto asignado a una variable

El primer mensaje "Nombre", permite visualizar en pantalla la palabra Nombre, Nom es la variable que almacenará el dato capturado por teclado.

Las variables son importantes, pero tanto como almacenar los datos en variables, también es importante el nombre que se le dé a cada una porque facilita el trabajo en el futuro.

En Python, dicha asignación por entrada por teclado sería así:

```
Nom = input("Escriba nombre") #Se le pide a usuario nombre
```

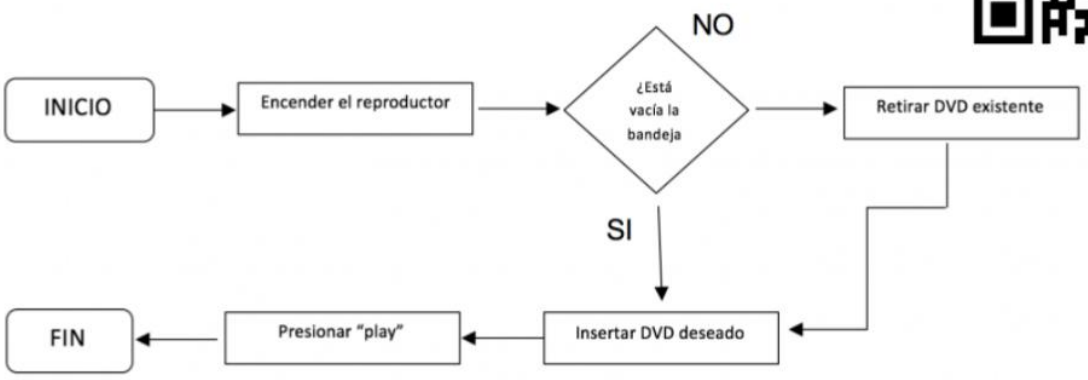


Recapitulando...

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Linea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida

Símbolo	Nombre	Función
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

Aplicando...



Última edición: 22 de diciembre de 2018. "Diagrama de flujo". Autor: María Estela Raffino. Para: Concepto.de. Disponible en: https://concepto.de/diagrama-de-flujo/. Consultado: 22 de julio de 2021.

Actividad no. 3
Nombre: _____

Instrucciones: A partir de lo revisado en las páginas previas con relación a los diagramas de flujo, elabora un diagrama de flujo para conectar un teléfono inteligente a una nueva red de internet, recuerda puedes hacer el uso del rombo para indicar procesos de decisión como en el ejemplo del DVD.

Diagramas de flujo, reglas a seguir.

Como se pudo observar en el material anterior, los diagramas tienen símbolos específicos para determinadas acciones, también hay un par de cosas que debemos cuidar con respecto a las líneas y notas que se incluyan en el diagrama, a continuación algunas importantes:

Reglas utilizadas en los diagramas

A continuación algunas reglas que facilitarán el diseño y validación de diagramas de flujo.

- Los diagramas de flujo se escriben de arriba abajo y de izquierda a Derecha.

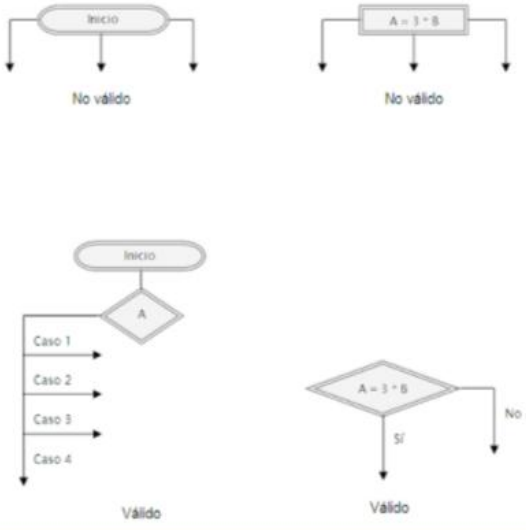
- Todo símbolo (excepto las líneas de flujo) llevará en su interior información que indique su función exacta y unívoca.

La manera en que se dirigen las líneas del diagrama conducen el flujo del lector y por lo tanto condicionan su entendimiento.

Reglas utilizadas en los diagramas

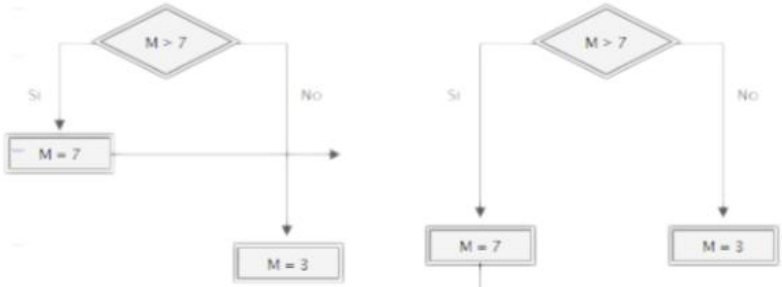
Reglas que facilitan el diseño y validación de diagramas de flujo.

- Un elemento del diagrama no puede tener más de una salida si no es un elemento de decisión. Existen excepciones que veremos en su debido momento...



Reglas utilizadas en los diagramas

Reglas que facilitan el diseño y validación de diagramas de flujo.



Las líneas de flujo no pueden cruzarse.
El ejemplo de la izquierda es INCORRECTO

No válido Válido

La inicialización de variables es fundamental, siempre que se requiera.
Lo anterior es muy común cuando requerimos introducir fórmulas.

Inicialización de variables

Es importante mencionar que las variables deben ser inicializadas en nuestros símbolos.

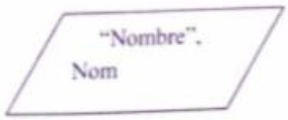
En los lenguajes de programación, normalmente lo anterior ocurre de forma automática para las variables locales aunque depende del tipo de dato. Sin embargo, para variables globales este comportamiento puede no ocurrir…

Es fundamental inicializar en diagramas de flujo.
A la derecha se aprecian variables de distintos tipos de datos con Python y debajo inicialización de valores en variables en un diagrama de flujo.

```
[ ]: # tipos de variables
[2]: type(1)
[2]: int
[5]: type(0.1)
[5]: float
[3]: type('texto')
[3]: str
[4]: type(True)
[4]: bool
```

Lectura de datos (símbolo: romboide)

- Ejemplo de entrada de datos con asignación de variable



Ejemplo de símbolos usados para entrada de datos

El primer mensaje "Nombre", permite visualizar en pantalla la palabra Nombre, Nom es la variable que almacenará el dato capturado por teclado.

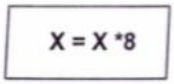
Para la lectura de datos se utilizan paralelogramos, en Python lo practicaremos más adelante con la función input()

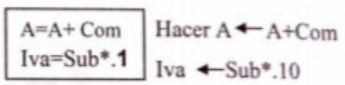
Operaciones aritméticas (símbolo: rectángulo)

- Como se ha observado de acuerdo a los tipos de datos y los operadores, se pueden llevar a cabo una gran diversidad de operaciones aritméticas: **dentro de las más comunes, podemos encontrar la suma, resta, multiplicación y división.**

  En diagramas de flujo se aplica dentro de la instrucción "hacer" (rectángulo, símbolo de proceso) la operación…

Ejemplos:



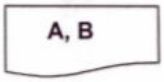


Salida de datos (símbolo: rectángulo irregular)

- En los diagramas de flujo, la salida de datos se representa por medio de un rectángulo con el lado inferior redondeado.

Impresión de datos

Ejemplo: Flujo para representar proyección de datos en diagrama

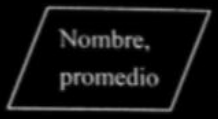
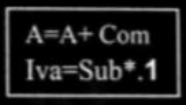
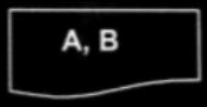


Escribir A, B

Con Python practicaremos más adelante con las salidas mediante el uso de la función print()

PYTHONIZATE
Aprende a programar sin perder la cabeza

Recapitulando lo anterior...

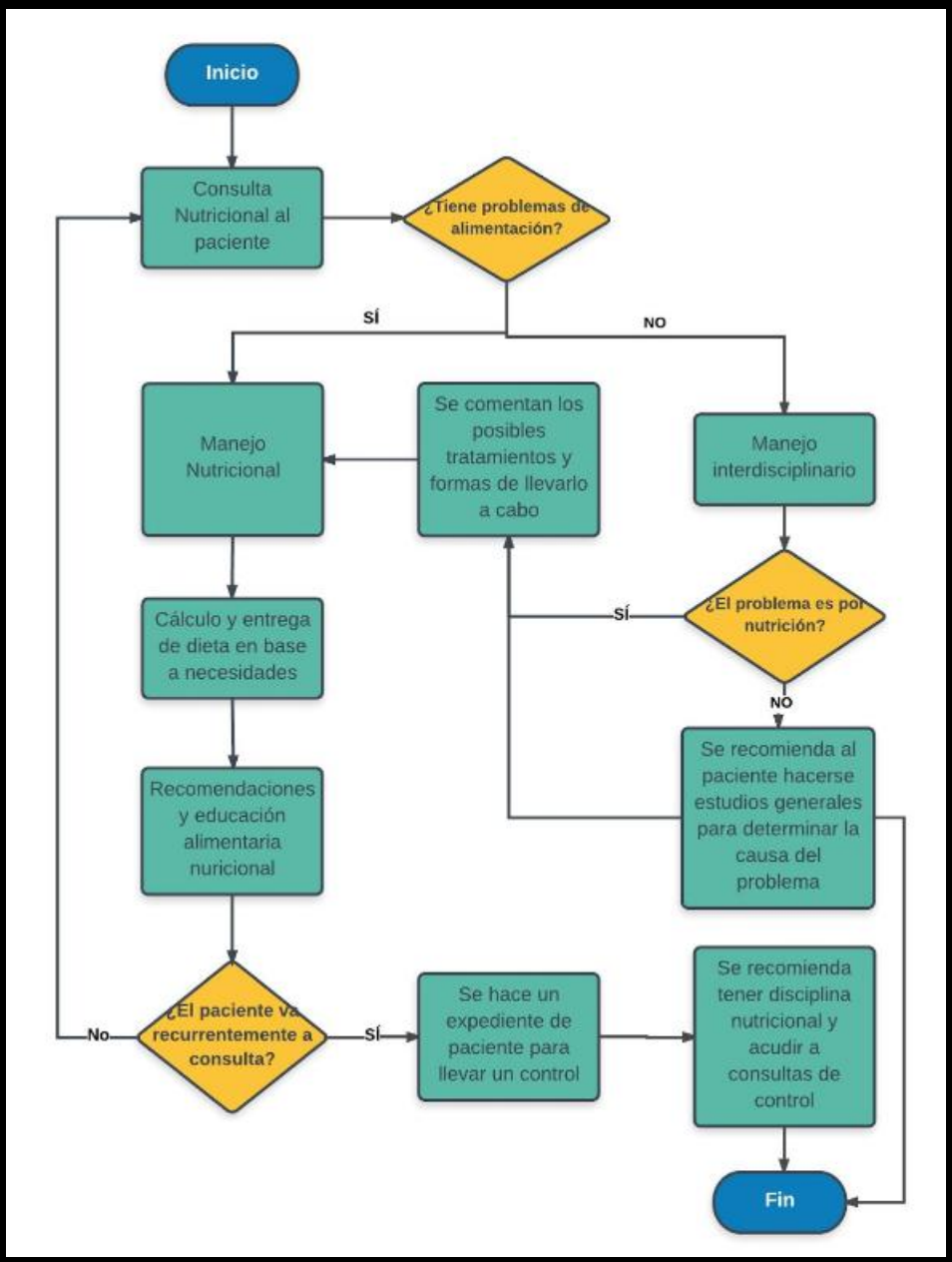
Simbolo	Instrucción	Ejemplo
	Leer…	 Leer nombre, promedio
	Hacer…	 Hacer A ← A+Com Iva ← Sub*.10
	Escribir...	 Escribir A, B

Para concluír esta sección teórica de introducción a los diagramas e inspirar a la práctica y a la reflexión sobre el tema se recomienda hacer algunos ejercicio con algún diagrama de flujo o también condicionarse a leerlos, hay muchos ejemplos a la hora de escribir en Google "ejemplo de diagrama de flujo de datos". También, algunos autores los refieren como flujogramas

Actividad no. 4
Nombre: _____

Instrucciones: Se presenta a continuación un diagrama de datos de un tema de salud alimentaria (los flujogramas sirven para todas las disciplinas). Indica el número total de caminos posibles que tiene el mismo. Indica también cuántas estructuras de decisión tiene.

También anota de manera libre, hacia el lado derecho de la página, todo lo que se consideras importante respecto al diseño, los símbolos que usa o los colores, todo aquello que resulta relevante para la interpretación del mismo.



Diagramas de flujo, estructuras de selección (estructuras condicionales).

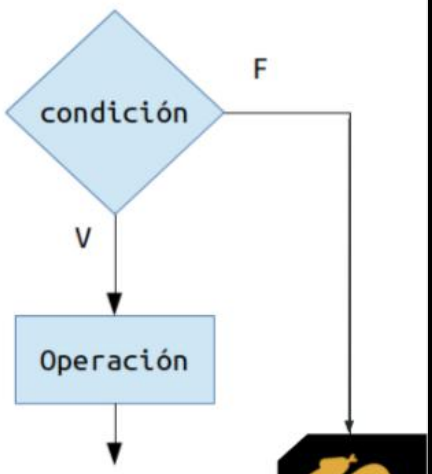
Como se pudo observar en el ejercicio previo cuando queremos “dividir” nuestro proceso hacia más posibles rutas es posible llevarlo a cabo, para ello empleamos estructuras de selección condicional.

Las estructuras lógicas selectivas se encuentran en la solución algorítmica de casi todo tipo de problemas. Las utilizamos cuando en el desarrollo de la solución de un problema debemos tomar una decisión, para establecer un proceso o señalar un camino alternativo a seguir, **la decisión sobre hacia dónde ir se sustenta en una condición lógica**, es decir una evaluación que solo puede ser o verdadera o falsa.

Estructuras de selección

La estructura si entonces permite que el flujo del diagrama siga un camino específico si se cumple una condición o un conjunto de condiciones.

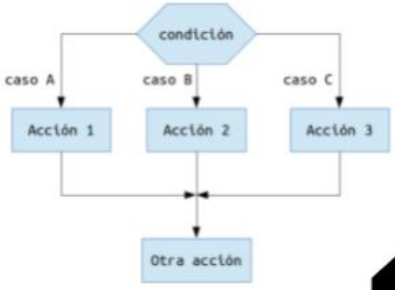
Si al evaluar la condición el resultado es verdadero se ejecuta(n) cierta(s) operación(es). Esta estructura se dice que es binaria dado que solo tiene dos posibles respuestas: si o no.



Estructuras de selección múltiple

La estructura selectiva permite que el flujo del diagrama se bifurque por varias ramas en el punto de la toma de decisión(es), esto en función del valor que tome el selector. El selector múltiple, a diferencia del selector simple, no es binario sino que tiene 3 o más posibles ramas para la decisión.

*Este tipo de selector permite elegir entre un número determinado de posibilidades, un número discreto (finito) de ellas. Por esta razón es que si se necesita evaluar un conjunto continuo (infinito) de posibles valores, este tipo de selector no es apropiado para ello.



Estructuras algorítmicas repetitivas

En la construcción de algoritmos para resolver un problema específico es muy común encontrarse con la necesidad de repetir alguna operación para la realización de esa tarea.

En dicha repetición, que a partir de ahora llamaremos iteración, puede ser necesario resolver alguna(s) operación(es) aritmética(s) o la evaluación de alguna decisión.

Cuando eso pasa decimos que se necesitan de las estructuras algorítmicas repetitivas.

PYTHONIZATE
Aprende a programar sin perder la cabeza

Una estructura repetitiva es útil para contar o acumular datos, como números o estructuras, en el caso de estructuras puede ser por ejemplo una lista con diversos datos.

Estructuras algorítmicas repetitivas

Todo ciclo debe terminar de ejecutarse luego de un número finito de iteraciones, por lo que es necesario verificar la(s) condición(es) en cada iteración para determinar si se debe continuar la ejecución o detenerse.

De esta forma todos los ciclos tienen estos tres elementos:

- **Condición Inicial**: Es el punto de partida del ciclo, debe ser una condición verdadera para que el ciclo pueda iniciar.
- **Condición Final**: Establece el punto final del ciclo, una vez que se cumpla la condición el ciclo deberá terminar.
- **Incremento/Decremento/Cambio**: Es el cambio de las variables que controlan el ciclo. El cambio puede ser un incremento, decremento o alguna otra condición.

Se verá a continuación qué son los contadores y acumuladores

Contadores y acumuladores

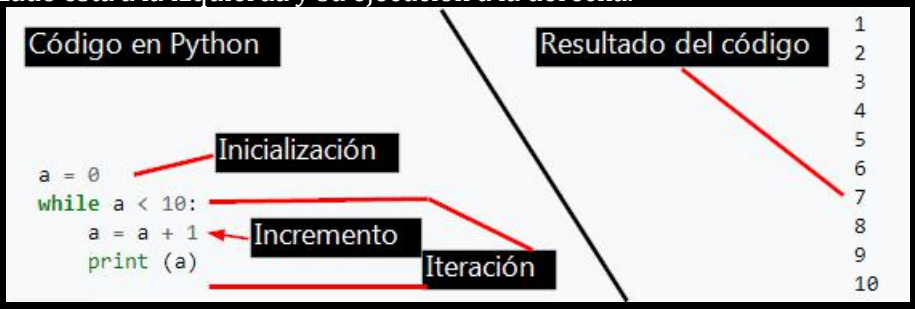
Un contador es un objeto que se utiliza para contar cualquier evento que pueda ocurrir dentro de un programa, en general de cuenta de forma natural, es decir desde 0 y de 1 en 1, aunque se puedan realizar otro tipo de cuentas.

Un contador no es otra cosa que una variable cuyo valor se incrementa o decrementa en una cantidad constante en cada repetición.

Las operaciones básicas que se realizan utilizando contadores son:

- Inicialización
- Incremento

Se presenta a continuación un ejemplo en Python, se volverá a esto más adelante, es un adelanto.
El código analizado está a la izquierda y su ejecución a la derecha.



Presentamos las operaciones básicas de los contadores y acumuladores a continuación:

Contadores

Las operaciones básicas que se realizan utilizando contadores son:

- Inicialización:
  Normalmente todo contador se inicializa en 0.
  - Ejemplo:
    contador ← 0
- Incremento:
  Cada vez que ocurre un evento que se desea contabilizar, se incrementa el contador en uno, si se realiza cuenta natural.
  - Ejemplo:
    contador ← contador + 1.

PYTHONIZATE
Aprende a programar sin perder la cabeza

Acumulador

Un acumulador es una variable que se utiliza en un programa para acumular elementos sucesivos con una misma operación.

El acumulador almacena cantidades variables resultantes de sumas sucesivas. Realiza la misma función que un contador sólo que el incremento o decremento de cada suma es variable en lugar de constante.

- Ejemplo:

    suma ← suma + número

Las operaciones básicas que se pueden realizar con un acumulador son:

- Inicialización
- Incremento

Acumuladores

Las operaciones básicas que se realizan utilizando acumuladores son:

- Inicialización:
  Es necesario iniciar el acumulador en un valor neutro de la operación que se va a acumular, en caso de la suma es 0, en caso de producto es 1.
  - Ejemplo:

      Acumulador ← 0
- Acumulación:
  Cuando el elemento a acumular aparece ya sea por la lectura de datos o por el cálculo, la acumulación del elemento se realiza por medio de la asignación:
  - Ejemplo:

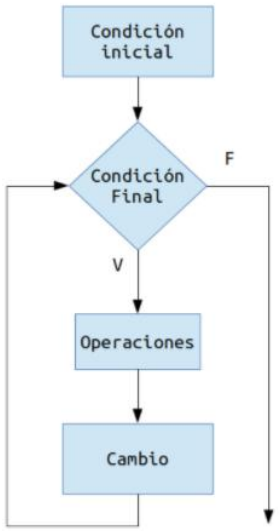
      acumulador ← acumulador + número.

También hay que reconocer que hay ocasiones en que no queremos hacer un incremento o acumulación pero no es útil repetir una serie de pasos en tanto ocurra o deje de ocurrir algo, para ello podemos contar con las estructuras repetitivas.

Estructuras algorítmicas repetitivas (mientras)

La estructura mientras conocida comúnmente como while, es la estructura algorítmica cuya característica principal es que **no se sabe de antemano el número de iteraciones que ocurrirán**.

Esta estructura también se encuentra presente en todos los lenguajes de programación.

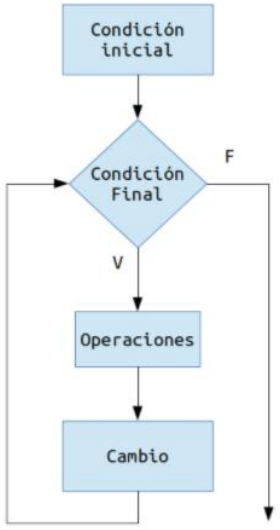


A cada grupo de pasos que ocurren en un estructura repetitiva le llamamos iteraciones y un conjunto de iteraciones forman un ciclo

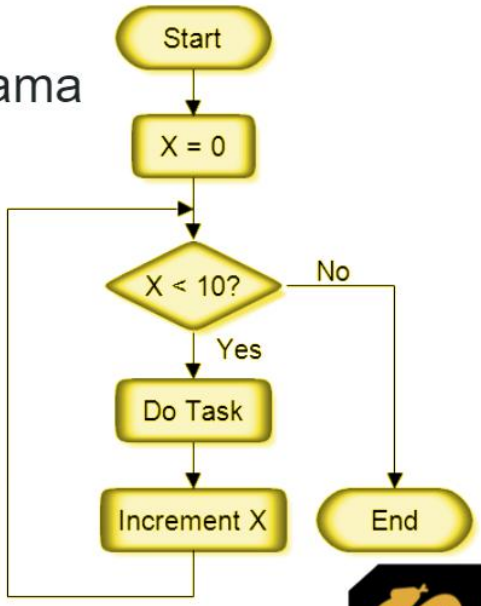
Estructuras algorítmicas repetitivas (mientras)

Para utilizar este ciclo se debe establecer un punto de partida y un final; es decir, una condición inicial y una condición final. El control de las iteraciones lo realiza una variable de control. ***Al conjunto de iteraciones también se le llama: ciclo.***
Para comenzar el ciclo la variable de control toma el valor de la ***condición inicial*** y en cada iteración se evalúa esta variable de control contra la ***condición final.*** Un ciclo ***realizará iteraciones mientras tal evaluación sea verdadera***.

No se sabe necesariamente con exactitud en estas estructuras cómo cambia la variable de control, lo que sí se debe saber con precisión es donde comienza el ciclo y dónde termina.

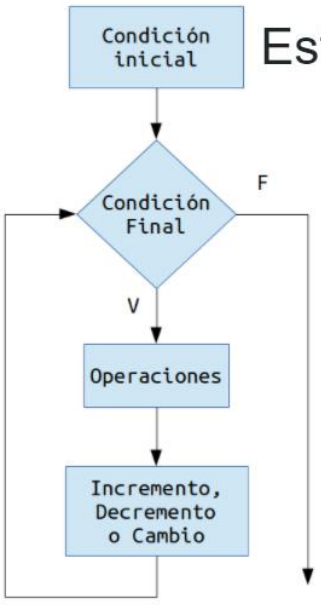


Contador con Python y su flujograma



Nota, hay un par de diferencias entre el código Python y el diagrama:
1.- La variable "x" se sustituyó por "a" en el código
2.- La operación "Do Task" en el código es un operación de salida (impresión) de datos con el valor de la variable "a" al momento de la iteración

Existe otra estructura repetitiva, denominada "para". En esencia cumple la misma funcionalidad de repetición aunque tiene una diferencia de origen en implica que sabremos cuantas veces se repetirá.

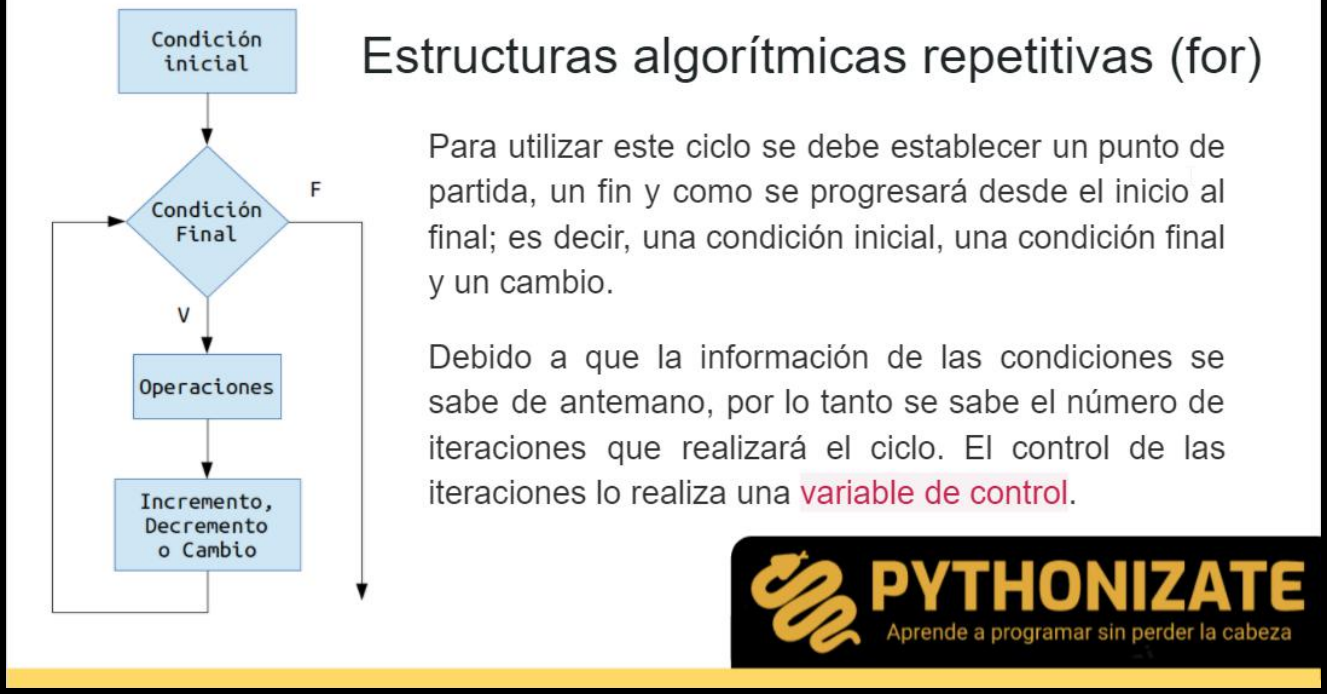


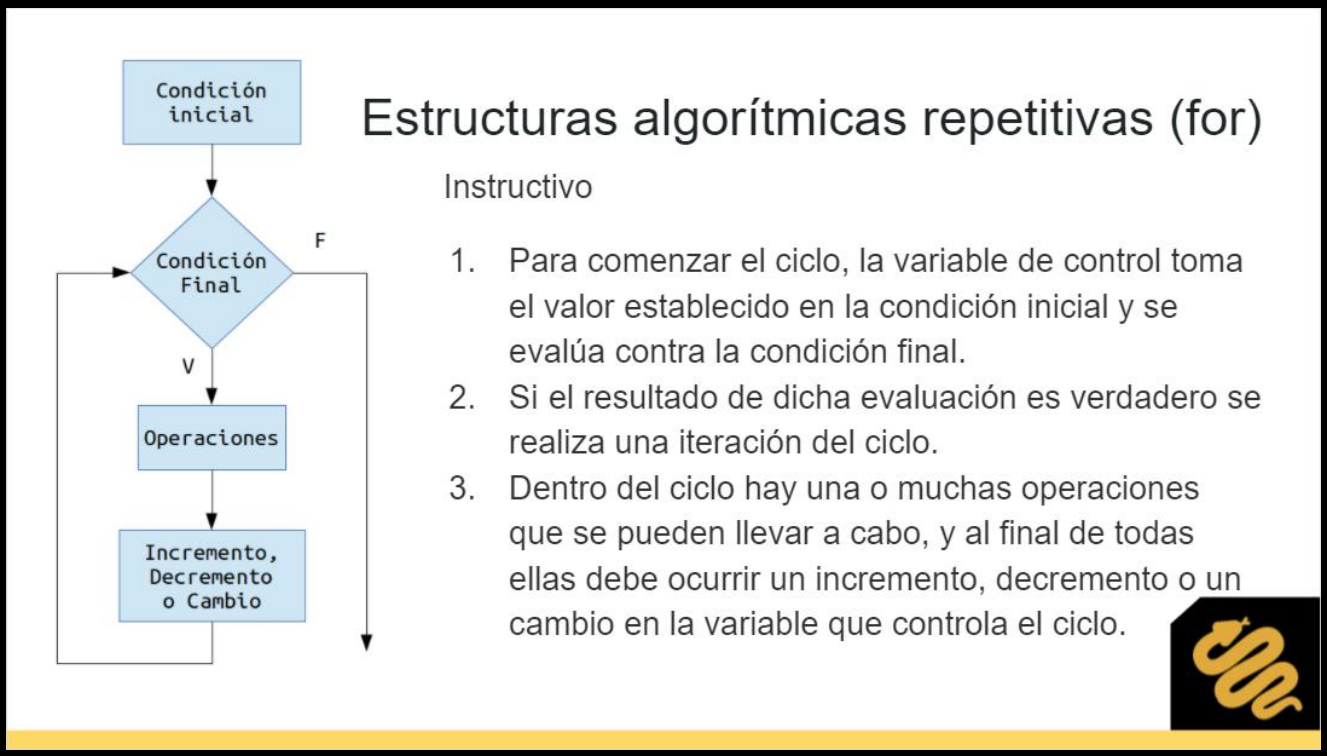
Estructuras algorítmicas repetitivas (para)

La estructura repetir conocida comúnmente como "para" (for, en inglés), es la estructura algorítmica cuya característica principal es que **se sabe de antemano el número de iteraciones que ocurrirán**

Esta estructura se encuentra presente en todos los lenguajes de programación.

PYTHONIZATE
Aprende a programar sin perder la cabeza



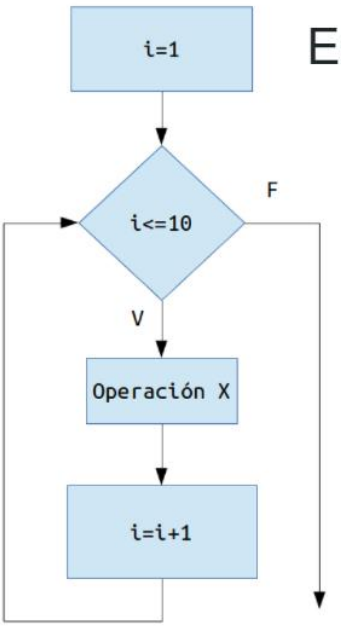


Repasando

Todo ciclo debe terminar de ejecutarse luego de un número finito de iteraciones, por lo que es necesario verificar la(s) condición(es) en cada iteración para determinar si se debe continuar la ejecución o detenerse.

De esta forma todos los ciclos tienen estos tres elementos:

- **Condición Inicial:** Es el punto de partida del ciclo, debe ser una condición verdadera para que el ciclo pueda iniciar.
- **Condición Final:** Establece el punto final del ciclo, una vez que se cumpla la condición el ciclo deberá terminar.
- **Incremento/Decremento/Cambio:** Es el cambio de las variables que controlan el ciclo. El cambio puede ser un incremento, decremento o alguna otra condición.



Estructuras algorítmicas repetitivas (for)

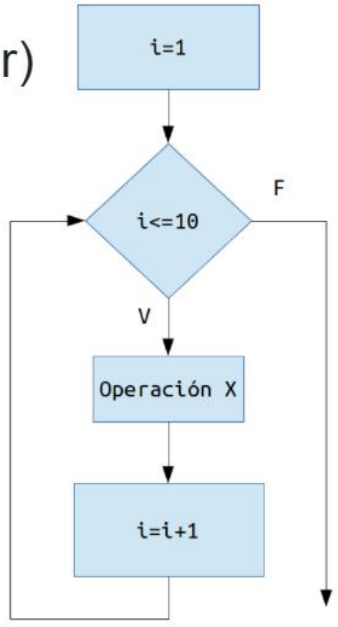
Descomponiendo el problema: Considerar que existe una operación X que necesita ser realizada 10 veces

1. Se define una variable de control. Digamos i
2. La variable i deberá contar el número de iteraciones.
3. Partiendo de lo anterior la condición inicial es i=1, la condición final i=10 y el cambio de la variable es un incremento en 1

Estructuras algorítmicas repetitivas (for)

Reflexionar el problema y sus implementaciones:

1. ¿Por qué se puso *i* <= 10 y no *i* != 10?
2. A diferencia de mientras, en el ciclo repetir se sabe necesariamente con exactitud cómo cambia la variable de control, conociendo dónde comienza el ciclo y dónde termina.



PYTHONIZATE
Aprende a programar sin perder la cabeza

A partir de este punto, el material del curso reduce la teoría y se centra más en ejercicios y el aprendizaje de lo básico del lenguaje Python

Reglas a seguir en Python

Ha llegado el momento para que comencemos a programar. Aquí hay unas sencillas normas que deben respetarse para una exitosa programación en Python. Aunque existen más, para fines prácticos del curso se resumen a ocho.

Al principio es probable que no entiendas nada, pero conforme vayas avanzando en este paquete didáctico todo cobrará sentido. Por ahora es importante que te familiarices y vuelvas para revisarlas cuando tengas dudas:

1. Los nombres de los ficheros deben escribirse en minúscula. Ejemplo:

   hola.py calculadora.py

2. Las llamadas a funciones se escriben en minúscula. Ejemplo:

   print() input()

3. Las variables se escriben en minúscula y, de estar formadas por varias palabras, éstas van unidas por guiones bajos. Ejemplo:

   balones piezas_de_repuesto

4. Los tipos de dato se escriben en minúscula. Ejemplo:

   str int

5. Pon un espacio después de cada coma al enumerar parámetros o concatenar. Ejemplo:

   ```
   print('Tienes', num_zapatos, 'zapatos y', num_blusas, 'blusas.')
   ```

6. Pon un espacio antes y después de cada operador. Ejemplo:

   8 + 2 total += 6

7. Indenta con 4 espacios; **nunca** uses el tabulador. Es muy importante que aprendas esto último. Incluso si escribes tu código a mano, respeta la regla. Ejemplo:

   ```
   if nombre == 'Ángel':
       print('Hola', nombre)
   1234
   input()
   ```

8. Escribe abundantes comentarios en tu código, describiendo cada detalle, para hacer que sea lo más claro y legible posible.

Salida de información

Recordando temas de sistemas vistos al inicio del curso, la salida de información implica que presentemos datos en algún medio, dispositivos que hacen uso de salidas de información todo el tiempo son las pantallas de tu teléfono, el monitor o pantalla de la PC o laptop, una impresora, etc. Los programas también hacen salidas de información a partir de la información que procesan.

Texto

Todo programa hace una serie de acciones básicas. Una de estas acciones es la de mostrar información: texto, números, resultados... es algo imprescindible.

Antes dijimos que un programa se componía de pequeñas líneas de código, órdenes simples. En adelante llamaremos a estas órdenes *funciones*.

Vamos a hacer nuestro primer programa. Este consistirá en mostrar un texto.

Para mostrar texto en Python usamos la función *print*, cuya sintaxis es:

```
print('texto')
```

Ejemplo en lenguaje humano:

Orden: Dime "Me llamo Sergio y tengo 16 años"
Respuesta: Me llamo Sergio y tengo 16 años

Lenguaje Python:

Orden: print('Me llamo Sergio y tengo 16 años')
Respuesta: Me llamo Sergio y tengo 16 años

Recuerda: *print('el texto que quieres mostrar')*

Comentarios

Los comentarios son anotaciones que hace el programador en su código fuente. Sirven para que al leer el código, otros programadores puedan entenderlo fácilmente (o incluso el autor original, al volver a leerlo pasado un tiempo).

En Python los comentarios se hacen con una almohadilla (#). Ejemplo:

```
# Solo afecta al texto que va a continuación de la almohadilla.
print('Hola') # Saludamos.
```

Es un buen hábito escribir comentarios y te conviene acostumbrarte a hacerlo.

Números

¿Y si en lugar de texto quiero mostrar una operación?

Es muy fácil, solo debes tener en cuenta una regla: cuando vas a mostrar texto, el interior de los paréntesis de *print* va con comillas simples:

print('texto')

Pero cuando queremos mostrar números u operaciones, va sin comillas:

print(1 + 2)

Se aprecia la diferencia, ¿verdad?. Ahora vamos a poner un ejemplo:

Ejemplo en lenguaje humano:

Orden: Dime, ¿cuánto son 10 menos 2 y medio?
Respuesta: Son 7 y medio.

Lenguaje Python:

Orden: print(10 – 2.5)
Respuesta: 7.5

Aquí solo hemos mostrado números pero, ¿podríamos mostrar texto y números al mismo tiempo en nuestro programa? La respuesta es que sí.

Orden: print('10 por 3 son', 10 * 3)
Respuesta: 10 por 3 son 30

Para combinar texto con operaciones debes respetar la regla antes mencionada: el texto lleva comillas, los números no. La frase *'10 por 3 son'* aunque contiene números, a estos se los considera texto y por eso llevan comillas.
*10 * 3* es una operación; no es texto y por eso no lleva comillas.

Importante: el texto va separado de los números y operaciones por coma.

Orden: print('Mi Windows tiene', 60 + 40, 'virus.')
Respuesta: Mi Windows tiene 100 virus.

Ponemos una coma a cada lado de la operación para separarlo de los dos textos.

Seguro que te habrás desconcertado un poco al ver que la multiplicación en Python se hace con asterisco (*) y no con equis (x). También te habrá sorprendido ver que en los números con parte fraccionaria no hay separación por coma, sino por punto; de tal manera que el número *6,30* en Python es *6.30*.

Te conviene conocer los tipos de número, así como las operaciones aritméticas que puedes hacer en este lenguaje (suma, resta, multiplicación...) y otras más potentes.

Tipos de número y operaciones:

En Python, dentro del sistema numérico decimal, podemos operar con números enteros y números reales. Los números enteros son aquellos que no tienen parte fraccionaria; y los números reales son los que sí la tienen. Ejemplo:

```
Número entero:    100
Número real:      20.30
```

Hay algo importante a tener en cuenta, y es que el resultado de una operación será entero o real dependiendo del tipo de números con los que se opere.
De esta manera, si todos los números son enteros el resultado también lo será; pero si al menos uno de los números fuera real, el resultado será real. Ejemplo:

```
20 + 30 = 50        # El resultado es entero, porque todos los números lo son.
5 * 2 = 10          # El resultado también es entero.

20.0 + 30 = 50.0    # El resultado es real, porque hay un número real.
5 * 2.5 = 12.5      # El resultado también es real.
2.2 – 0.2 = 2.0     # El resultado también se muestra como real.
```

He aquí una tabla que muestra algunas de las operaciones básicas que pueden hacerse en Python:

Operación	Operador	Ejemplo
Suma	+	2 + 4 = 6
Resta	–	2 – 4 = -2
Multiplicación	*	3 * 3 = 9
Potencia	**	3 ** 3 = 27
División	/	50 / 6 = 8.3
Cociente	//	50 // 6 = 8
Resto	%	50 % 6 = 2

Nota: asumo que el lector tiene conocimientos de matemáticas suficientes como para entender lo que se expresa en la anterior tabla.

Variables

Ya sabemos cómo mostrar información y operar con números, pero hasta ahora nuestra información era estática y no se guardaba en ningún lado.

Una *variable* es un elemento que permite almacenar información. Para comprender su funcionamiento pondré como ejemplo una comparación.

Imagina que tienes 2 cajones y cada uno lleva una etiqueta puesta. Dentro de cada cajón tendremos guardado algo:

```
nombre = 'Almudena'
```

De esta manera nuestro programa tendrá control en todo momento sobre los datos que manejamos. Si Almudena cumpliera un año más, su edad se incrementaría:

```
edad = edad + 1
```

Pongamos un ejemplo en Python:

```
nombre = 'Almudena' # Guardamos el nombre de la chica.
edad = 19 # Guardamos su edad.

# Mostramos su nombre:
print('Te llamas', nombre)

# A continuación mostramos cuántos años tiene:
print('Tienes', edad, 'años.')

edad = edad + 1 # Le agregamos un año

# Volvemos a mostrar su edad:
print('Cumples', edad, 'años ¡felicidades!')

input() # Hacemos pausa hasta que el usuario pulse la tecla INTRO.
```

El programa mostraría lo siguiente:

```
Te llamas Almudena
Tienes 19 años.
Cumples 20 años ¡felicidades!
```

Te habrás percatado de que la función *print* también sirve para mostrar variables. Las variables también deben respetar la norma que vimos antes: van sin comillas y deben estar separadas del texto por coma.

La variable debe su nombre a que la información que guarda puede variar.
Para manejar la información podemos usar los operadores que vimos en la anterior tabla; y además, operar con varias variables a la vez. Ejemplo:

```
comida_niños = 50 # Hay 50 bandejas de comida para niños.
comida_adultos = 30 # Hay 30 bandejas de comida para adultos.

# Mostramos las bandejas disponibles:
print('Tienes', comida_niños, ' bandejas de comida para niños.')
print('Tienes', comida_adultos, ' bandejas de comida para adultos.')

# Simulamos que los clientes han hecho consumos:
comida_niños = comida_niños – 40
comida_adultos = comida_adultos – 20

# Sumamos las bandejas y las guardamos en otra variable:
total_comida = comida_niños + comida_adultos

print('Ha habido venta. En total quedan', total_comida, 'bandejas de comida.')

input() # Hacemos pausa hasta que el usuario pulse la tecla INTRO.
```

El programa mostraría lo siguiente:

```
Tienes 50 bandejas de comida para niños.
Tienes 30 bandejas de comida para adultos.
Ha habido venta. En total quedan 20 bandejas de comida.
```

Importante: en el caso de *"comida_niños = comida_niños – 40"* estamos haciendo un decremento en el valor de la variable, pero de esta manera queda demasiado largo. Podría hacerse lo mismo de una manera más corta, pero sólo es válido cuando interviene una variable y un número; o dos variables. Ejemplo:

```
comida_niños = comida_niños – 40 # Método largo.
comida_niños –= 40 # Método corto. Hace lo mismo que el largo.
```

Estaríamos haciendo lo mismo, pero ahorrando espacio y tiempo.
Aquí otro ejemplo, esta vez con dos variables:

```
total = 10
sobrante = 20
total += sobrante # Sumamos el contenido de "sobrante" a la variable "total".

print('El total mas el sobrante es', total)

input() # Haemos pausa hasta que el usuario pulse la tecla INTRO.
```

El programa mostraría lo siguiente: *El total mas el sobrante es 30*

Ahora te enseñaremos lo que **nunca** debes hacer:

```
cds_reggaeton = 2
cds_metal = 7
total = cds_reggaeton += cds_metal # Sólo pueden intervenir dos variables.

# Esto daría un error sintáctico. No se puede mezclar el metal con reggaeton…
# No es broma
```

He aquí una tabla en la que podrás ver cada operación con su correspondiente ejemplo simplificado:

Operación	Ejemplo	Ejemplo simplificado
Asignación	a = b	a = b
Suma	a = a + b	a += b
Resta	a = a – b	a -= b
Multiplicación	a = a * b	a *= b
Potencia	a = a ** b	a **= b
División	a = a / b	a /= b
Cociente	a = a // b	a //= b
Resto	a = a % b	a %= b

Entrada de información

Otra acción imprescindible que realiza un programa es la de tomar información. Esto le permite interactuar con el usuario, adaptándose a las necesidades de éste. Si un programa no toma información los resultados siempre serán los mismos, por lo que resultará de poca utilidad.

La función input()

Ya habrás visto esta función en ejemplos de código anteriores. Nosotros la usaremos para dos cosas: hacer pausas en nuestro programa y permitir que el usuario introduzca información.

Veamos un ejemplo de pausa:

```
input() #Código en alguna parte del programa que lo pausa
```

Aquí el programa se quedaría esperando hasta que el usuario pulse INTRO.

También podemos hacer que se muestre un mensaje junto con la función. Ejemplo:

```
input('Pulsa INTRO para continuar...')
```

El programa mostraría lo siguiente:

```
Pulsa INTRO para continuar...
```

Tras pulsar INTRO el programa continuaría con su ejecución o, en caso de ser *input()* la última línea de código, el programa finalizaría de forma normal.

Ahora que ya sabemos cómo hacer pausas con esta función, vamos a aprender a pedirle información al usuario y guardarla en variables. Para ello conoceremos algunos tipos de datos, los cuales se muestran en la siguiente tabla:

Tipo	Clase	Ejemplo
str	Texto	'Esto es un texto'
int	Número entero	23
float	Número real	3.14

La información de la tabla anterior puede parecer algo confusa al principio, pero ahora te lo explicaré detalladamente para que lo entiendas:

```
nombre = str (input('Introduce tu nombre'))
print('Te llamas', nombre)
```

Usamos *str o cadena* cuando queremos que el usuario introduzca cualquier tipo de texto como palabras, frases, etc. Cabe destacar que también podemos operar con texto (sumarlo, restarlo, multiplicarlo...), por increíble que te pueda parecer. Ejemplo:

```
# Este programa se inventa palabras compuestas:
palabra_1 = str (input('Introduce la primera palabra'))
palabra_2 = str (input('Introduce la segunda palabra'))

compuesta = palabra_1 + palabra_2 # Hacemos unión de ambos textos.

print('Me he inventado esta palabra:', compuesta)
```

El tipo *int o entero* se utiliza cuando queremos que el usuario introduzca números enteros. Este tipo también es válido para números muy grandes. Ejemplo:

```
edad = int (input('Introduce tu edad'))
print('Tienes', edad, 'años.')
```

Otro ejemplo:

```
dist_pluto = int (input('¿A cuántos kilómetros está Plutón de la Tierra?'))
print('Está a', dist_pluto, 'kilómetros.')
```

Usaremos el tipo *float o flotante* para que el usuario introduzca números reales. Ejemplo:

```
pi = float (input('¿Cuál es el número PI?')) # Ya sabemos que es 3.14
print('El número PI es', pi)
```

Y con esto tenemos lo necesario para crear una interacción básica con el usuario.

Con lo que se ha visto hasta ahora has acumulado una cantidad considerable de información y ya va siendo hora de ir asimilándola. ¿Qué mejor que unos cuantos ejercicios para ponerte a prueba?

Nombre: _____

Actividad no. 5

Nota: se entiende que todos los ejercicios deberán estar comentados y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programay su respectivo diagrama de flujo…

1. **Muestre el texto "Hola mundo".**

2. **Pida al usuario su nombre y edad, y los muestre.**

3. **Pida al usuario dos números enteros, los sume y muestre el resultado.**

Actividad no. 6

Nota: se entiende que todos los ejercicios deberán estar comentados y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programa y su respectivo diagrama de flujo, que:

1. **Calcule el área de un círculo. Será el usuario quien introduzca el radio. La fórmula es *PI* multiplicado por *radio* al cuadrado ($3.14 \cdot r^2$).**
2. **Pida al usuario un número real y calcule su raíz cuadrada. Para esto puedes usar la propiedad de las potencias ($\sqrt{n} = n0.5$).**

Condiciones

Las condiciones dotan de inteligencia a nuestros programas. Usando una condición puedes plantear una igualdad que, si se cumple, conlleva a una determinada acción.

Para comprobar una igualdad usamos los siguientes operadores relacionales:

Operación	Operador	Ejemplo	Descripción
Igual	==	a == b	Comprueba si *a* y *b* son iguales.
Desigual	!=	a != b	Comprueba si *a* y *b* sin distintos.
Mayor que	>	a > b	Comprueba si *a* es mayor que *b*.
Menor que	<	a < b	Comprueba si *a* es menor que *b*.
Mayor o igual a	>=	a >= b	Comprueba si *a* es mayor o igual a *b*.
Menor o igual a	<=	a <= b	Comprueba si *a* es menor o igual a *b*.

A continuación se explicará cómo usar condiciones.

if - else

Bien, *if* y *else* son dos funciones que pueden funcionar en conjunto o de manera independiente para crear condiciones. Pongamos un ejemplo con *if*:

```
nombre = str (input('¿Cómo te llamas?')) # Pedimos nombre al usuario

# Establecemos condición: si el usuario se llama Cristian, saludamos.
if nombre == 'Cristian':
    print('Hola', nombre)
1234

input('Pulse INTRO para finalizar...') # Pausamos.
```

Se ha establecido que, si el usuario se llama Cristian, el programa le saludará. En caso de no cumplirse esto, el programa terminaría sin mostrar nada.

Algo que hay que señalar y es muy importante: para que una serie de acciones estén sujetas a una condición deberás indentarlas con 4 espacios, **nunca** con tabulador. Todo lo que no esté sujeto a una condición va sin indentado.

Como ya se abordó, se puede usar *if* de forma independiente. En cambio, para usar *else* **siempre** debe haber antes de éste un *if*.

El efecto de *else* es que si no se cumple la condición de *if*, entonces el bloque de acciones de *else* se ejecuta. Pongamos otro ejemplo para verlo con más claridad:

```
# Pedimos  al usuario que introduzca su nombre.
nombre = str (input('¿Cómo te llamas?'))

# Comprobamos si el usuario se llama Laura.
# Siempre que se cumpla if, el else nunca se ejecutará.
if nombre == 'Laura':
    print('Hola', nombre)
```

```
# Si no se cumple la condición de if, entonces sí se ejecuta el else.
else:
    print('No te conozco.')
```

```
# El input está afuera de ambas condiciones, así que siempre se ejecutará.
# Para dejar afuera un bloque, éste debe estar sin indentar (sin espacios).
input('Pulse INTRO para finalizar...') # Hago una pausa.
```

Cabe destacar que después de cada llamada a *if* y *else* llevan dos puntos (:).

Ahora supongamos que queremos comprobar varias condiciones, ¿podríamos poner varios *if* uno detrás de otro? la respuesta es que sí. Ejemplo:

```
# Este programa comprueba si cumplimos los requisitos de Ubuntu 12.04.
# Pedimos información sobre el procesador:
procesador = float (input('¿Cuántos megaherzios tiene su procesador?'))
# Pedimos información sobre la RAM:
ram = int (input('¿Cuánta memoria RAM tiene su ordenador?'))

if procesador >= 1000: # Comprobamos potencia del procesador.
    print('Tu procesador cumple los requisitos.')
```

```
else:
    print('Tu procesador no cumple los requisitos.')
```

```
if ram >= 1024: # Compruebo la cantidad de RAM.
    print('Tienes suficiente memoria RAM.')
```

```
else:
    print('No tienes suficiente memoria RAM.')
```

```
input('Pulse INTRO para finalizar...') # Pausamos
```

elif

Hay otro método para hacer varias comprobaciones, aunque el efecto es distinto. Si queremos hacer varias comprobaciones y que varios bloques de acciones se ejecuten, entonces usaremos varios *if*; pero si queremos hacer varias comprobaciones y que sólo se ejecute un bloque de acciones, usaremos *elif*.

```
# Este programa comprueba si un número es positivo o negativo:
num = int (input('Introduzca un número')) # Pido un número al usuario.

if num < 0:
   print(num, 'es un número negativo.')
elif num > 0:
   print(num, 'es un número positivo.')
elif num == 0:
   print(num, 'no pertenece a ningún grupo.')

input('Pulse INTRO para finalizar...') # Hacemos una pausa.
```

También podemos añadir un *else* después del útimo *elif* para contemplar otros posibles sucesos. Ejemplo:

```
nombre = str(input('¿Cómo se llama usted?')) # Pedimos nombre.

# El programa comprobará si te llamas Alberto o Regina.
# De cumplirse te saludará, pero si tienes otro nombre dirá que no te conoce.
if nombre == 'Alberto':
   print('Hola', nombre)
elif nombre == 'Regina'
   print('Hola', nombre)
else:
   print('No te conozco.')

input('Pulse INTRO para finalizar...') # Hacemos una pausa.
```

Importante: esto no es más que una cuestión de gustos. Dependiendo de tu estilo de programación (y del propósito de tu programa) te sentirás más cómodo con un método u otro, pero lo cierto es que *elif* ofrece muchas facilidades.

Para poder usar *elif*, antes debe existir un *if*. Puedes combinar *if-elif-else*, pero *else* sólo podrá intervenir una vez, después del último *elif*, como muestra el ejemplo.

Nombre: _____

Actividad no. 7

Ahora que has aprendido a usar condiciones es momento de ponerlo en práctica para terminar de asimilarlo. En la página siguiente encontrarás las soluciones.

Nota: se entiende que todos los ejercicios deberán estar comentados y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programa y su respectivo diagrama de flujo, que:

1. Pida dos números reales, compruebe cuál es mayor y lo muestre.

2. Pida tu edad y compruebe si tienes o no la mayoría de edad. En caso de introducir un número menor que cero, el programa devolverá un mensaje de error.

Actividad no. 8

Nota: se entiende que todos los ejercicios deberán estar comentados y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programa y su respectivo diagrama de flujo, que:

1. **Pida un número entero, la nota final de una asignatura. Si es menor que 0, devolverá un mensaje de error; si es menor o igual que 5, mostará por pantalla "reprobado"; si es igual a 6, "insuficiente"; si es menor a 8, "aprobado"; si es igual o si es menor que 9, "notable"; y si es mayor o igual a 9, "sobresaliente".**

Bucles

El bucle es otro tipo de condición, por así llamarlo, permite la repetición de un evento de forma controlada. Con él también podemos plantear una igualdad, con la diferencia de que mientras ésta se cumpla (o mientras no se cumpla, recordando que evalúa una condición con una salida lógica tipo verdadero/falso), el bloque de acciones sujeto al bucle se irá repitiendo una y otra vez. El bucle sólo finalizará cuando la condición que la acciona ya no corresponda.

A continuación se explicará cómo usar el bucle *while*.

while

El bucle *while* se puede utilizar cuando desconocemos el número de veces que se repetirá un bloque y para el *manejo de excepciones*, entre otras cosas.

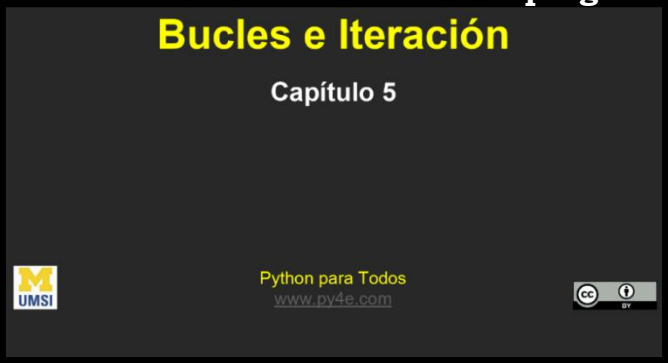
Veamos un ejemplo de repetición de un bloque:

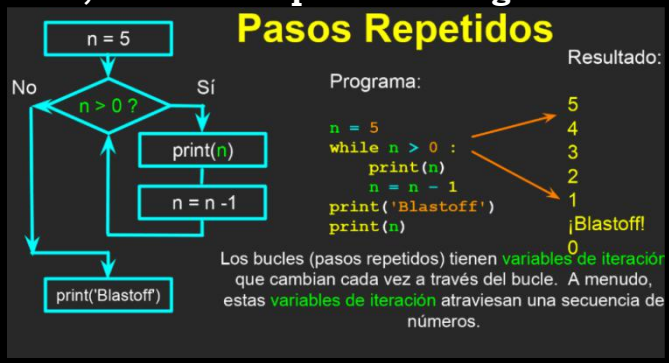
```
# Este programa nos servirá para comprar un número dado de cómics:
comics_usuario = 0 # Número de cómics que tiene actualmente el usuario.
# Preguntamos al usuario cuántos cómics le gustaría tener.
num_comics = int (input('¿Cuántos cómics te gustaría tener?'))

# Mientras tengamos menos cómics de los que te gustaría tener...
while comics_usuario < num_comics:
    print('Tienes', comics_usuario, 'comics.')
    print('Has comprado un cómic nuevo.')
    comics_usuario += 1
    input('Pulse INTRO para continuar...') # Hacemos una pausa.
1234

input('Pulse INTRO para finalizar...') # Hacemos otra pausa.
```

El *manejo de excepciones* sirve para impedir que ocurran imprevistos en la ejecución de un programa. Por ejemplo, cuando quiero que el usuario introduzca sólo números mayores que cero.

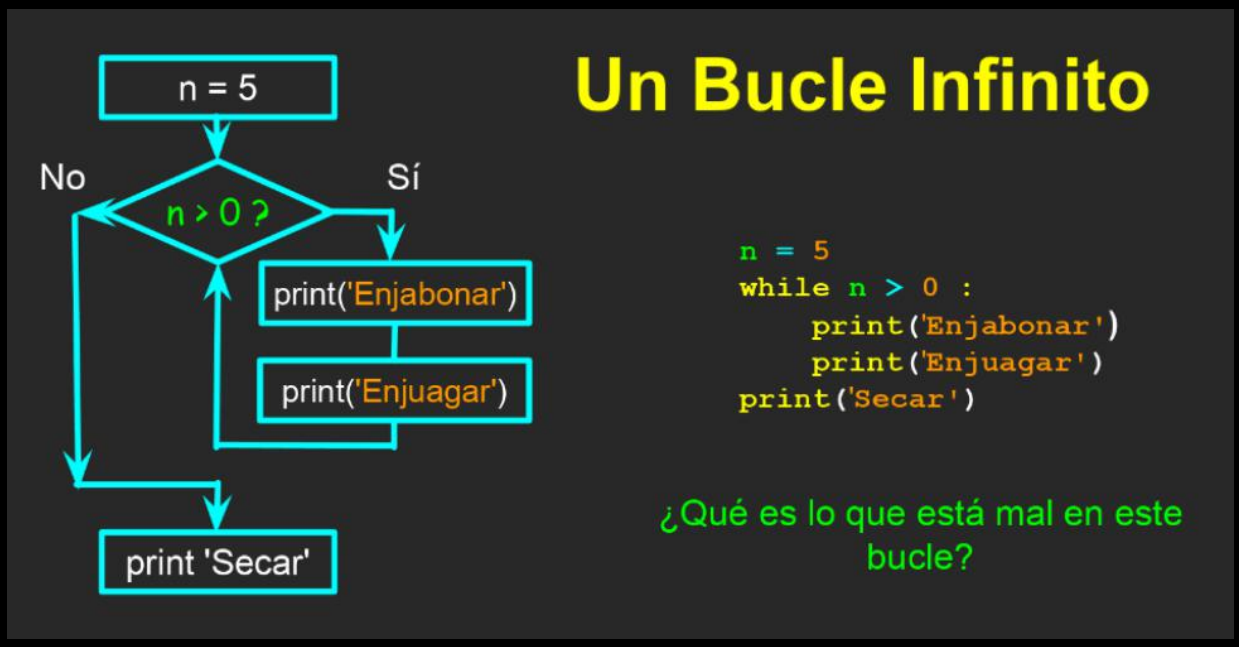
```
# Preguntamos al usuario cuántos cómics le gustaría tener.
num_comics = int (input('¿Cuántos cómics te gustaría tener?'))

while num_comics <= 0: # Evitamos una excepción
    print('Debes introducir un número mayor que cero.') # Advertencia.
    # Hacemos que el usuario introduzca otro número.
    # Si sigue siendo menor o igual a cero, el bloque volverá a repetirse.
    num_comics = int (input('¿Cuántos cómics te gustaría tener?'))
1234

input('Pulse INTRO para finalizar...') # Hacemos una pausa.
```

A continuación se presentan algunos ejemplos de Bucles presentados en el curso de Charles Severance de programación, con sus respectivos diagramas

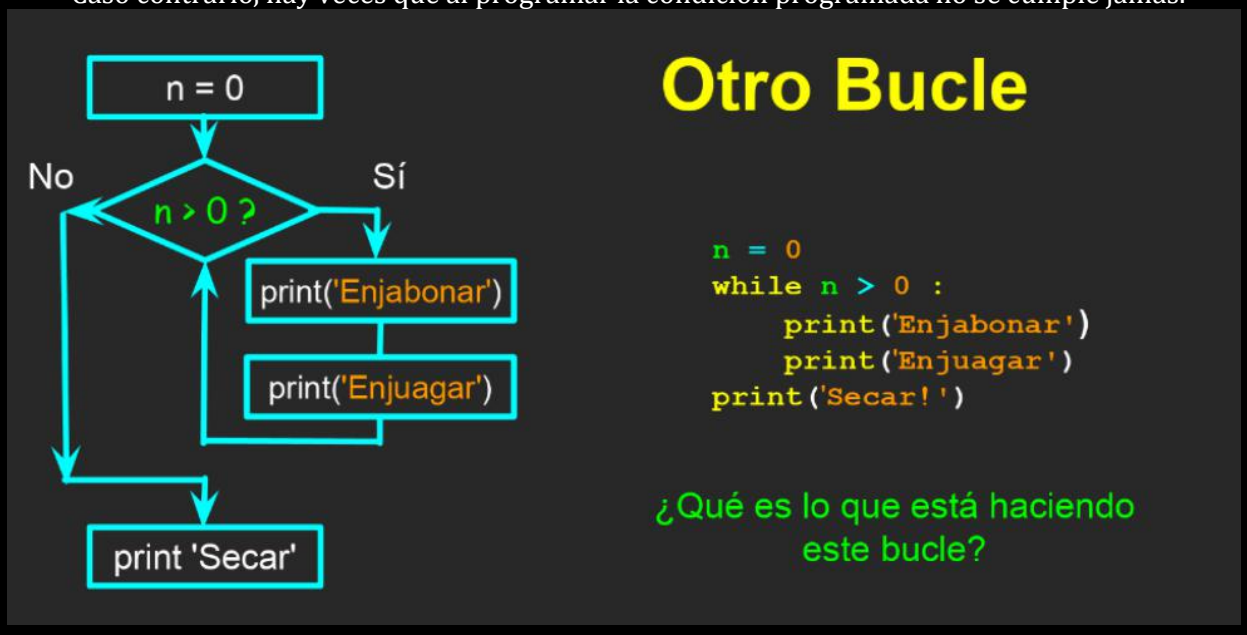




En ocasiones cuando no tenemos cuidado al programa nuestros ciclos, las cosas se pueden salir de control, en el proceso de análisis del algoritmo, debemos asegurar que no haya escenarios en que el ciclo se siga ejecutando por no lograr alcanzar una condición que evalúe verdadero, como el ejemplo siguente:



Caso contrario, hay veces que al programar la condición programada no se cumple jamás.



Actividad no. 9

Nombre: _____

Ahora que sabes usar bucles (en teoría) es el momento de ponerlo en práctica.

Nota: el ejercicio deberá estar comentado; tener manejo de excepciones y hacer una pausa hasta que el usuario pulse INTRO, antes de finalizar.

Haz un programa y su respectivo diagrama de flujo, que:

1. Pida al usuario un número entero, impidiendo que éste sea menor que cero. El programa mostrará la tabla de multiplicar de dicho número, hasta el multiplicador 20 del mismo.

Los arreglos, una introducción a las estructuras de datos

- Los arreglos o arrays, por su nombre en inglés se refieren a ciertas estructuras de datos de tipo estático que son usadas típicamente para agrupar objetos del mismo tipo (aunque pueden ser distintos tipos también). Los arreglos dan la posibilidad de referirnos a un grupo de objetos a partir de un nombre en común.

- Cabe mencionar que las estructuras de datos estáticas son aquellas que quedan establecidas desde el momento en que ocupan espacio en memoria, es decir desde su creación, por lo que generalmente permanecen inmutables a lo largo de su ciclo de vida.

A los arreglos se les conoce también como matriz, vector o formación.

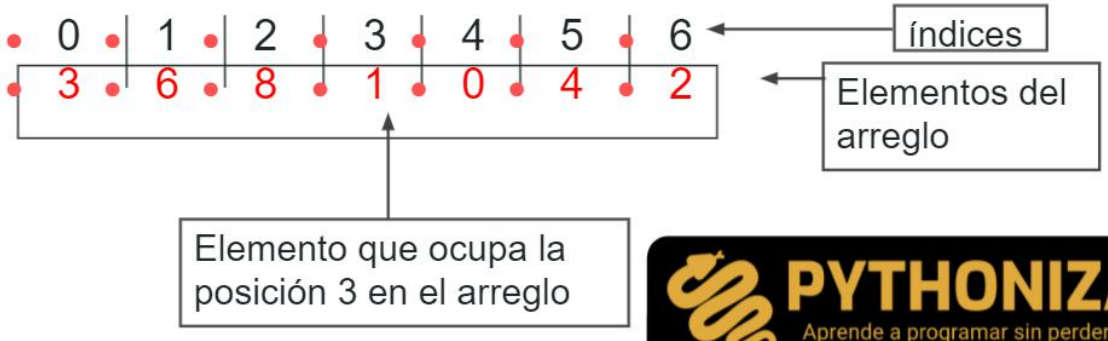
Los arreglos pueden ser:

- Unidimensionales
- Multidimensionales

Las matrices multidimensionales existen porque para cada dimensión de un arreglo unidimensional se le puede asignar a su vez un nuevo arreglo. Es decir que los arreglos multidimensionales son arreglos de arreglos

Definición simple

- Un arreglo es una estructura homogénea, compuesta por varios elementos, todos del mismo tipo y almacenados consecutivamente en memoria, cada espacio de almacenamiento en memoria constituye un índice.



PYTHONIZATE
Aprende a programar sin perder la cabeza

Declarar un arreglo con Python como se verá es sencillo

Declaración de un arreglo

- Para declarar un arreglo en Python, no necesitamos hacer mención del tipo de dato, solamente reservar un nombre para el arreglo en cuestión, siempre acompañado de los corchetes **"[ ]"** después de la asignación. Por ejemplo:

```
f = []
```

- También tenemos la ocasión de usar rangos para llenarlos en caso que los busquemos inicializar con un valor

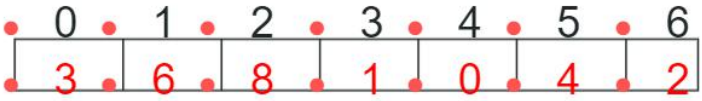
```
f = []

for i in range(30):
    f.append(0)
```

PYTHONIZATE
Aprende a programar sin perder la cabeza

Lo que a veces resulta un poco contraintuitivo es como acceder a un elemento que lo componen

Accediendo a elementos del arreglo

- Para acceder a un elemento de un arreglo, se coloca el nombre del arreglo y entre corchetes el índice del elemento que se desea.
- Por ejemplo: si se desea obtener el número 4 del siguiente vector, entonces…



main.py

```
1 miArreglo = [3,6,8,1,0,4,2]
2 valor = miArreglo[5]
3 print("Resultado: {}".format(valor))
```

Powered by trinket
Resultado: 4

Utilización y manipulación

- En todos los lenguajes de programación los arreglos se manipulan en base al segmento de espacio que ocupan, también conocido como índice.
- Los índices en un arreglo, van desde 0 hasta el tamaño n – 1.
- En Python el tamaño de un arreglo se obtiene con la función len(arreglo) que ya existe en el propio lenguaje, con la siguiente sintaxis:

```
totalElementos = len(arr)
```

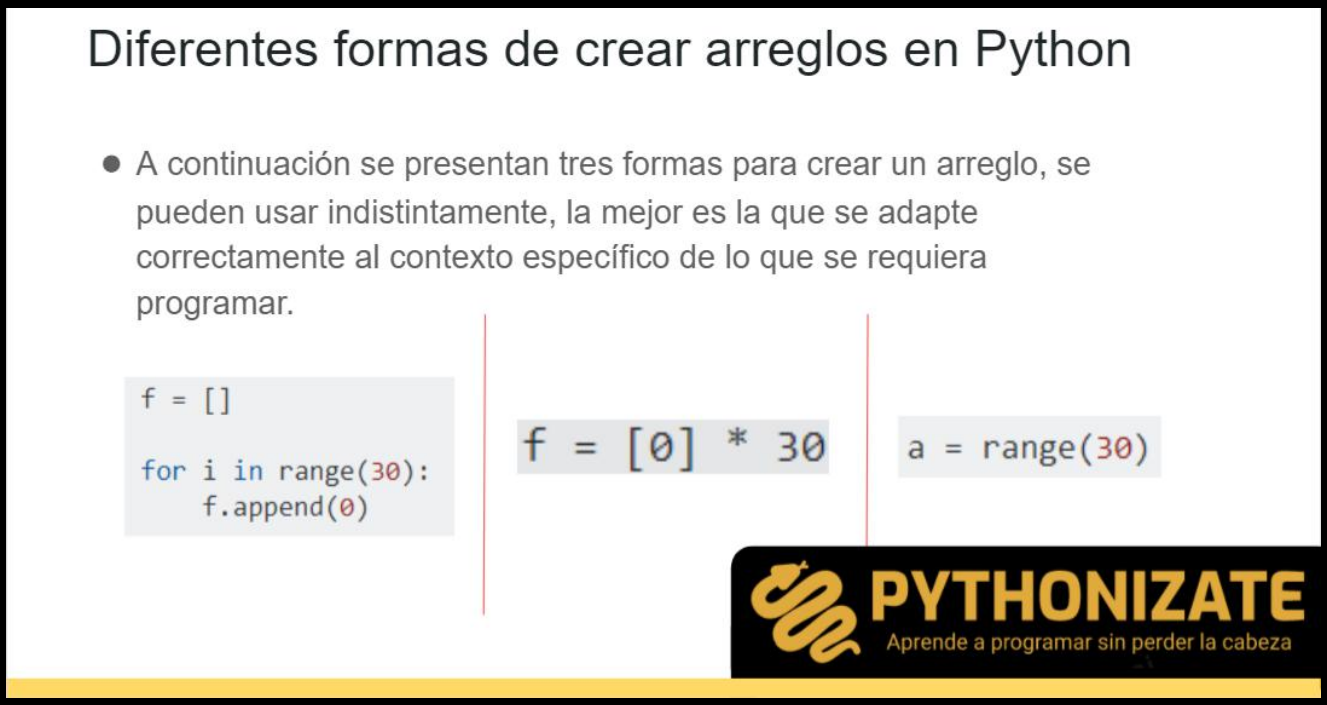
Analizado lo fundamental de la teoría de arreglos vamos a ver los en código Python

Ejemplos de operaciones con arreglos

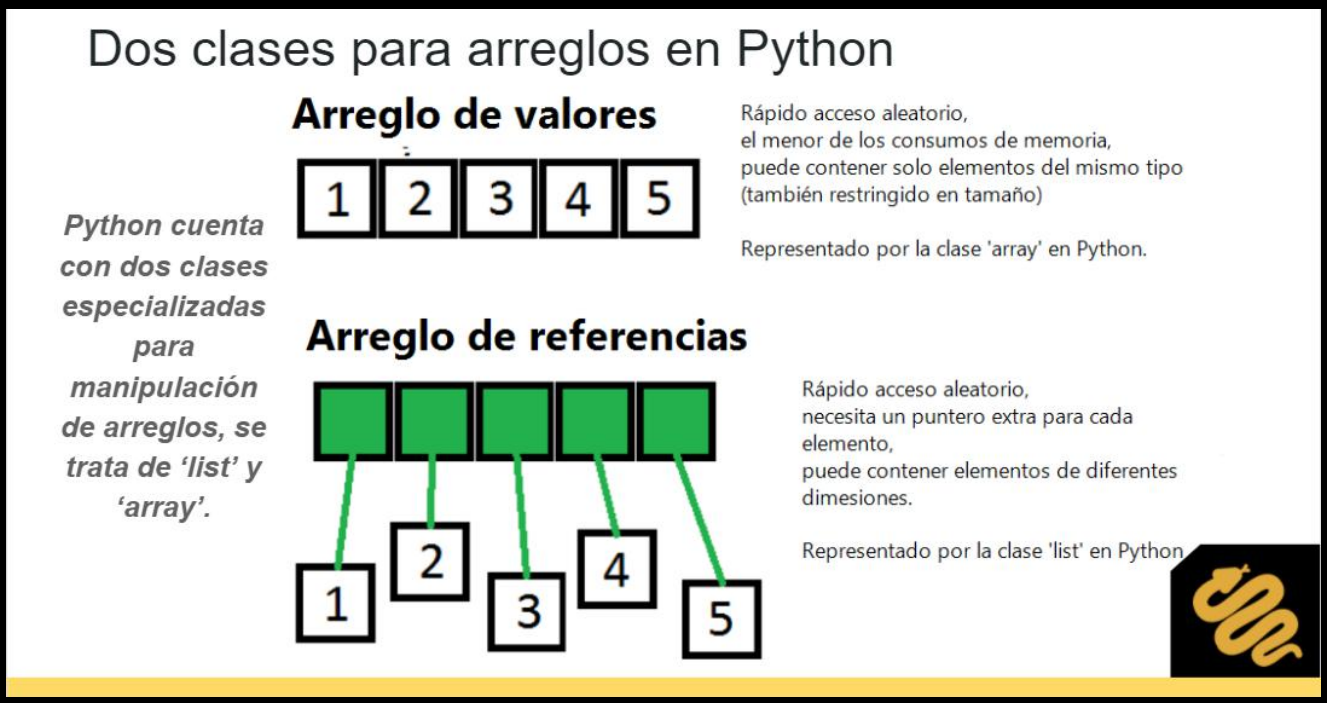
- En el siguiente ejemplo se despliegan operaciones posibles para arreglos

```
# arreglo vacio
arr = []

# inicializacion con arreglos (puede contener diferentes tipos de datos)
arr = [1, "eels"]

# obtener elemento por indice (indice negativo es para acceder a ultimo elemento)
arr = [1, 2, 3, 4, 5, 6]
arr[0]  # 1
arr[-1] # 6

# obtener longitud
length = len(arr)

# tambien se puede usar append e insert para agregar nuevos elementos
arr.append(8)
arr.insert(6, 7)
```

Como se comentaba, en Python y en otros lenguajes se puede realizar una acción de muchas formas diferentes, ejemplo:



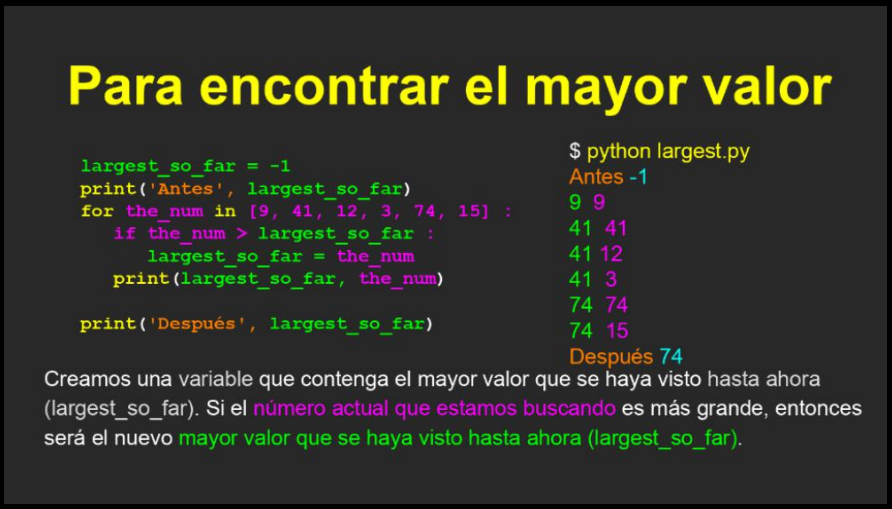
Ahora, distinguiremos que hay arreglos para números y arreglos que solo almacenan referencias.



Cuando hablamos de clases nativas del lenguaje, nos referimos a implementaciones, código preprogramado que podemos usar para escribir menos

Ejemplo adicional:
Lograr a partir de un arreglo identificar el valor mayor de entre una lista de números.
Lo anterior se lograr a partir de la comparación entre los diferentes elementos de la lista de número, como se observa en la diapositiva.

Tomado de los materiales de Python para todos del libro en español de Charles Severance:
https://es.py4e.com/



Actividad no. 10

Nombre: _____

Ahora que sabes usar arreglos (partiendo de la teoría) es el momento de ponerlo en práctica.

Realiza los siguientes programas con Python (en esta ocasión no se requieren los diagramas de flujo):

1. **Tenemos la siguiente lista de números [454,578,65,367,908,267,595,32,756,885]**
   Se nos solicita que separemos los números por grupos. Generen un código que nos permita sumarlos mediante dos grupos, uno que agrupe los que son pares de la lista y otro que agrupe a los nones. La función o código debe retornar las suma de los elementos pares y por otra parte también la suma de los elementos nones.

2. **Generar una funcionalidad que pida valores pares positivos al usuario: se deben almacenar solo los primeros diez. Si el usuario introduce valores nones o negativos se deben ignorar. Los valores 5to. y 6to. se deben igualar a cero y al final, se deben mostrar (imprimir) los datos que se hayan recabado para esos diez números cuando ya se tengan todos los datos.**

Actividad no. 11

Nombre: _____

Observa el siguiente código y responde las preguntas, notarás una nueva palabra no conocida, "class", esta hace referencia a una clase, una abstracción para la construcción de un objeto, las clases representan el pilar de la programación orientada a objetos, en la línea 3 podrás notar que es una función, aunque es una función especial conocida como constructor. Documentación muy amplia sobre el tema se puede encontrar en la documentación de Python en español: https://docs.python.org/es/3/tutorial/classes.html

```
1  # Se define una clase
2  class MyClase(object):
3      def __init__(self, number):
4          self.numero = number
5
6  # Se define un arreglo
7  my_objects = []
8
9  # definimos un contador
10 indice = 0
11
12 # Se agregan los valores enteros al arreglo
13 # y otras cosas mas
14 for i in range(100):
15     my_objects.append(MyClase(i))
16
17 # Se imprime valor de atributo x de
18 # elemento en la 10ma. posicion (no indice)
19 x = 0
20 print my_objects[x].numero
```

Pregunta 1: Seleccione la respuesta correcta
¿Qué valor debe tener x en la línea 19, para que la línea 20 imprima 51?

A) 50 **B) 51** **C) No es posible ese resultado**

Pregunta 2: Seleccione la respuesta correcta del código anterior
¿la variable de la línea 10, se declara y se usa en el código?

A) Verdadero **B) Falso**

Pregunta 3: Seleccione la respuesta que genera error al imprimir valores del siguiente arreglo bidimencional (es pregunta trampa porque ese tipo de arreglo no se presentó)

```
arregloBidimensional = [ [10,5], [23,9] ]
```

A) arregloBidimensional[-1][-1] **B) arregloBidimensional[0][1]**
C) arregloBidimensional[2][1] **D) Ninguno de los anteriores**

Pregunta 4: Genere un código que represente los números de una tirada de dos dados comunes (dado cúbico, los clásicos con seis caras)
Pista: este problema puede tener que ver con un arreglo bidimendional que son arreglos de arreglos, agrega aleatoriedad para que se pueda imprimir un resultado distinto por "tirada" (es decir, en cada ejecución del código).
La respuesta a esta pregunta esconde el inicio de una lección adicional

Bibliografía y agradecimientos

En el presente documento se extiende un particular agradecimiento a José Miguel Ruiz Torres por generar una guía en línea paso a paso para la instalación de Python en los diversos sistemas operativos, material que se remezcló de su formato original, tomando fragmento de los temas de bucles, condicionantes, operadores, entradas y salidas, así como del proceso de instalación para los diversos sistemas operativos.

Igualmente se incluyen las fuentes de la clase que dieron lugar a al material de las diapositivas que se comparte en la parte teórica sobre diagramas de flujo, teoría de sistemas y arreglos, entre otros.

Esperanza Manrique Rojas (2009). Empezando a programar. ILCSA, de donde se sintetizó parte de la lección de diagramas de flujo.

Héctor Julián Selley Rojas. (2020). Diagrama de Flujo. 24 julio 2021, de Universidad Anáhuac México Norte. Sitio web:
https://lab.anahuac.mx/~hselley/ayp/diagramasDeFlujo.html

Charles Severance. (2020). Repositorio CC Python para Todos, GitHub. 24 julio 2021, de Python para todos. Sitio web:
https://github.com/csev-es/py4e/tree/master/lectures3

Listas y Arreglos en Python, diferentes declaraciones en arreglos, recuperado de los foros de StackOverFlow
https://stackoverflow.com/questions/1514553/how-to-declare-an-array-in-python

Notas adicionales para lectores:

Si te interesa seguir tu camino en la programación y llevar tus habilidades a otro nivel se recomiendan mucho los materiales del curso de Python para todos (Py4E), en tanto no adaptemos una versión de ese material, realizado por el Dr. Charles Severance de la Universidad de Michigan, el curso tiene una edición parcialmente traducida al español y sus videos tienen subtítulos, conforme avanza se exploran temas más avanzados como estructuras de datos y algoritmos de ordenación, entre otros. Puedes encontrar los materiales en Coursera o en la página oficial:
https://es.py4e.com/
https://es.coursera.org/courses?query=charles%20severance

Notas adicionales para ampliar esta obra:

Si te interesa tomar esta guía y remezclarla puedes ver la licencia Creative Commons para que generes tu propia versión de este material.

https://creativecommons.org/licenses/by-nc-sa/3.0/es/

Por último, también te animamos a unirte a nuestro equipo de voluntarios escribiendo un correo a: ali@pythonizate.com