TEMA 1:

Fundamentos de la Programación

ÍNDICE

Final de estos apartados:

EJEMPLOS ILUSTRATIVOS
TRABAJO PERSONAL

1.1 Introducción (Conceptos básicos)

Objetivos

Introducir conceptos y métodos básicos de la informática utilizando para ello programación estructurada.

Informática: "Conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento informático de la información por medio de ordenadores (RAE, 2015)."

Programación Estructurada:" Forma de escribir programas de ordenador de forma clara. Para ello utiliza únicamente tres estructuras: secuencia, selección e iteración".

1.1 Introducción (Conceptos básicos)

Informática

- ***Informática teórica***: Estrechamente relacionada con la matemática aplicada. Comprende disciplinas como la teoría de autómatas, de lenguajes formales, de lenguajes recursivos, de la complejidad...

- ***Informática técnica***: Enfocada a la construcción de sistemas informáticos. Estrechamente relacionada con la electrónica. Se encarga de disciplinas como los circuitos, microprogramación...

Nosotros nos centraremos en la llamada:

- Informática práctica: Desarrolla métodos generales para la programación de sistemas y construcción de programas de servicio para los sistemas informáticos. Disciplinas que engloba:

  Sistemas operativos

  Lenguajes de programación y traductores,

1.1 Introducción (Conceptos básicos)

Conceptos claves Informática práctica

- Representar datos.
- Realizar procesamiento de datos mediante algoritmos para obtener resultados.

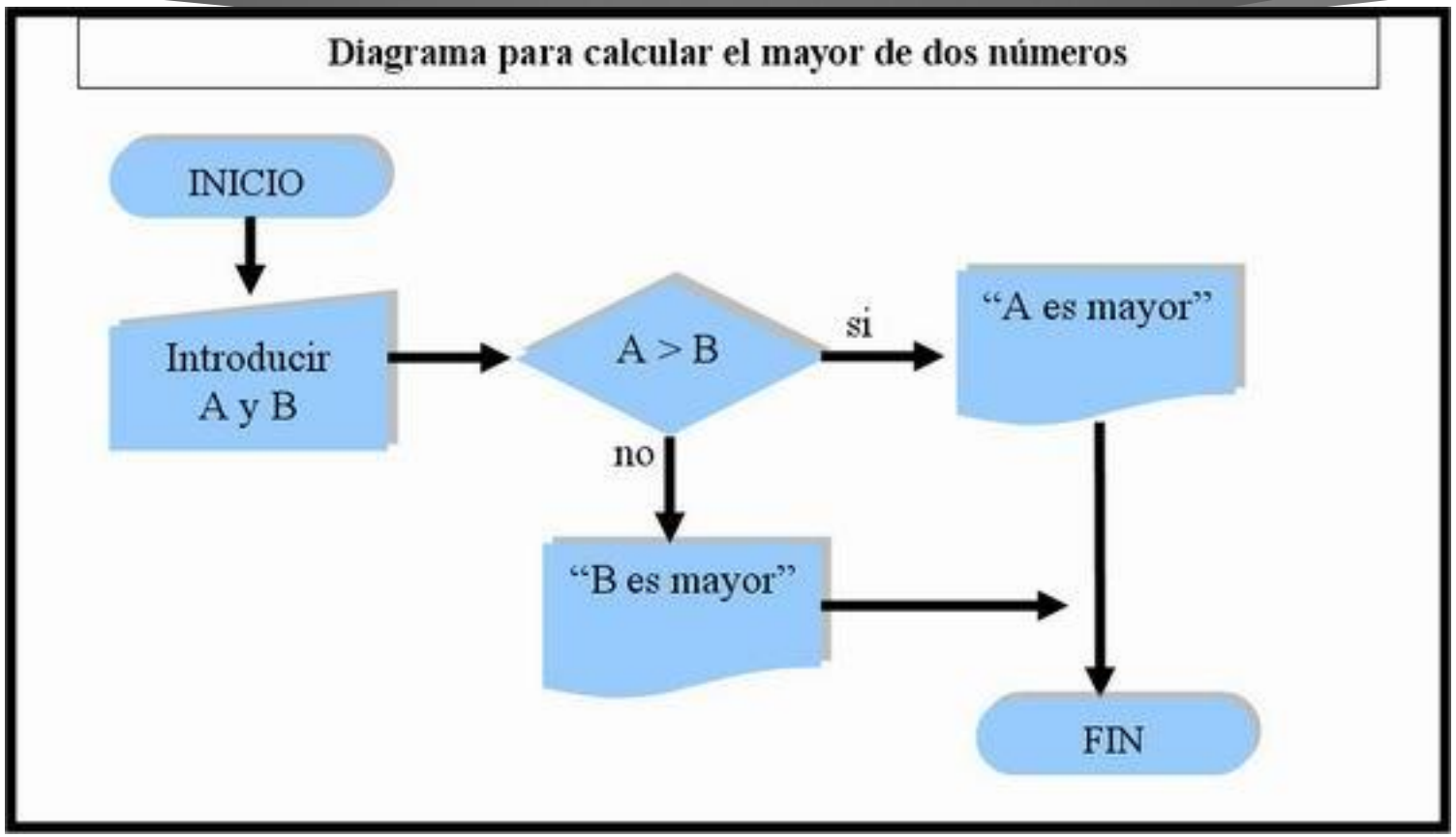
Algoritmo: *es una lista bien definida, ordenada y finita de instrucciones que permite resolver un problema*

1.1 Introducción (Conceptos básicos)

Características Algoritmos:

- Son ***precisos,*** es decir, constan de una serie de reglas que detallan paso a paso lo que hay que hacer.
- Forman un ***método sistemático y definido***, es decir, no dejan nada a la interpretación o improvisación, lo que implica que partiendo de las mismas premisas siempre obtiene los mismos resultados.
- Tienen un ***objetivo***, ya sea un problema a resolver o un resultado a obtener.
- Existe un cierto ***orden en la aplicación de las reglas***, necesario para alcanzar el objetivo en un número finito de pasos, de lo que se deduce que tienen un inicio y un fin.

1.1 Introducción (Conceptos básicos)



Representación mediante diagrama de flujo u organigramas

1.1 Introducción (Conceptos básicos)

Pasos a seguir para desarrollar un algoritmo

Análisis: Tener una definición exacta de lo que se quiere hacer.

Escribir algoritmo: Después de haber analizado el problema en una forma abstracta, se debe llevar al papel mediante instrucciones adecuadas al análisis. Si el problema fue bien analizado, este paso es muy rápido a comparación del anterior.

Prueba de Escritorio: Paso opcional, se aplica siguiendo paso por paso las instrucciones del algoritmo, anotando los diferentes valores que van tomando las variables, de forma que se pueda verificar si hay errores en alguna instrucción.

1.1 Introducción (Conceptos básicos)

PSEUDOCÓDIGO

Definición: Lenguaje general, de especificación de la solución del problema mediante un algoritmo.

Se considera el paso anterior a la codificación.

Ventajas: Este lenguaje puede ser traducido indistintamente y sin problemas a cualquier lenguaje de programación de computadora.

En esta asignatura se aprenderá a programar de forma estructurada. El lenguaje utilizado será Python, los conceptos que no se puedan simular con este lenguaje se implementarán en pseudocódigo.

NOTA: Python, es un lenguaje interpretado orientado a objetos, en esta asignatura solamente aprenderemos su sintaxis, intentando simular como trabaja un lenguaje estructurado. No se verá realmente como trabaja Python (POO) al final de la asignatura se mostrará un resumen de como funciona realmente.

1.1 Introducción (Conceptos básicos)

¿Qué es Python?

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses *Monty Python*. Caracerísticas:

- **Interpretado:** No se debe compilar el código antes de su ejecución. Un lenguaje interpretado es aquel en el cual sus instrucciones o más bien el código fuente, escrito por el programador en un lenguaje de alto nivel, es traducido por el interprete a un lenguaje entendible para la máquina paso a paso, instrucción por instrucción. El proceso se repite cada vez que se ejecuta el programa el código en cuestión. Los lenguajes interpretados permiten el tipado dinámico de datos, es decir, no es necesario inicializar una variable con determinado tipo de dato sino que esta puede cambiar su tipo en condición al dato que almacene entre otras características más.
- **Interactivo** : Python dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudarnos a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.
- **Orientado a Objetos** : La programación orientada a objetos está soportada en Python y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables.

Toda la información necesaria sobre el lenguaje la puedes encontrar en http://www.python.org/

1.1 Introducción (Conceptos básicos)

Ventajas de uso:

- La sintaxis es muy sencilla y cercana al lenguaje natural, lo que facilita tanto la escritura como la lectura de los programas.
- Es un lenguaje muy expresivo que da lugar a programas compactos, bastante más cortos que sus equivalentes en otros lenguajes.
- Es un lenguaje de programación multiparadigma, que permite al programador elegir entre varios estilos: programación orientada a objetos, programación imperativa (y modular) y programación funcional.
- Es multiplataforma por lo que podremos utilizarlo tanto en Unix/Linux, Mac/OS o Microsoft Windows.
- Es un lenguaje interpretado y por tanto interactivo. En el entorno de programación de Python se pueden introducir sentencias que se ejecutan y producen un resultado visible, que puede ayudarnos a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.
- Python es gratuito, incluso para propósitos empresariales.
- Es un lenguaje que está creciendo en popularidad. Algunas empresas que utilizan Python son Yahoo, Google, Disney, la NASA, Red Hat, etc.

1.1 Introducción (Conceptos básicos)

Comentarios de un programa

- Elementos vitales de nuestros programas.
- Ayudan a su comprensión, modificación y corrección.
- Pueden contener cualquier elemento y no se toma como parte del código del programa
- Python:

Comentario 1 línea:

```
a = 3 #esta variable toma el valor entero 3
```

Comentario varias líneas: se puede hacer de una sola vez seleccionando con el ratón las líneas a comentar y pulsando la combinación de teclas Alt+3. Cada vez que se pulsa Alt+3, se añaden dos almohadillas a cada línea

```
##esto es para probar
##los comentarios de varias lineas
print('hola')
```

Comentario docstring: Se estudiará en Tema 2, sirve para describir las funciones creadas

```
'''Esta llamada me permite calcular senos, cosenos, logaritmos, raíces  cuadradas,  etc..'''
```

1.1 Introducción (Conceptos básicos)

PROGRAMA

Un programa *"es la realización práctica de un algoritmo en un lenguaje de programación concreto" ó "una secuencia de acciones (instrucciones) que manipulan un conjunto de objetos (datos)".*

1.2. Estructura de un Programa

Partes de un Programa

1.4 Bloque Identificativo

(Cabecera Programa)

Cabecera Programa: Sirve para identificar lo que hace cada programa en Python no existe la cabecera pero nosotros escribiremos unos comentarios informativos del programa

1.5 Bloque declarativo y de definición

El bloque declarativo constituye la parte estática de todo programa donde se establece la declaración de datos o elementos que reciben la acción del programa. Python no se declararan

1.6 Bloque ejecutivo

(Cuerpo del Programa)

Bloque donde se establece la acción del programa sobre los datos determinados en el **bloque ejecutivo**, es la **parte activa**, claramente identificable con el contenido del algoritmo que se implementa a través del programa.
VER GUIA DE ESTILO OFICIAL DE PYTHON

1.3. Identificadores y Palabras reservadas

Identificadores

1. Nombre que se da a un elemento del programa, sirve para representarlo y hacer referencia a él.
2. Tienen unas reglas de formación específicas en cada lenguaje .

Reglas en PYTHON:

- Un identificador comienza con una letra o con guión bajo (_) y luego sigue con una secuencia de letras, números y guiones bajos. Los espacios no están permitidos dentro de los identificadores. Pueden contener caracteres no ingleses, como acentos, eñes, c cedillas, etc (a partir de Python3. La versión que instalaremos es la versión 3.4.2)
- Python distingue mayúsculas de minúsculas, así que Hola es un identificador y hola es otro identificador.

  Ejemplos: año, hola12t, _hola

1.3. Identificadores y Palabras reservadas

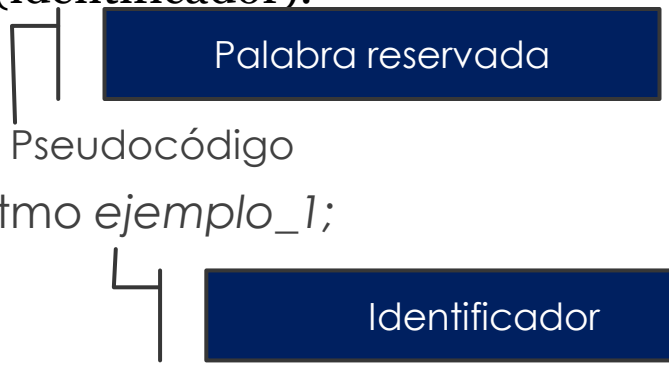
Palabras reservadas

1. Palabras que un lenguaje de programación tiene asignadas para realizar ciertas funciones.
2. No debe utilizarse palabras reservadas como identificadores.
3. Palabras reservadas en Python3:

and, as, assert, break, class, continue, def, del, elif,
else, except, finally, for, from, global, if, import, in, is,
lambda, nonlocal, not, or, pass, raise, return,
try, while, with, yield

1.4. Bloque identificativo

Cabecera de un programa, consta de una palabra reservada que identifica lo que vamos a comenzar a hacer + el nombre con el que queramos identificar nuestro algoritmo (identificador).



- Ejemplo Pseudocódigo

  Algoritmo *ejemplo_1;*

- Ejemplo Python:

  ## Archivo: XXXX_XX.py
  ## Autor: XXXXX XXXXXXX (nombre y apellidos)
  ## Fecha: DD/MM/AAAA
  ## Descripción: Ejercicio XXXXXXX. Este programa bla bla bla
  # #Si la **descripción es larga**, escriba varias líneas

1.5. Bloque declarativo

1.5. Bloque Declarativo

- 1.5.1. Módulos o bibliotecas.
- 1.5.2. Datos y Tipos de Datos.
- 1.5.3. Constantes.
- 1.5.4. Variables.
- 1.5.5. Subprogramas (procedimientos y Funciones)

1.5. Bloque declarativo

- El bloque declarativo constituye la parte estática de todo programa donde se establece la declaración de datos o elementos que reciben la acción del programa.
- En general en la mayoría de los lenguajes de programación estructurados después de la cabecera se enumeran y definen:
  - Las librerías que se emplearan (Trozos de código ya definidos con funciones que podremos emplear)
  - Datos y el tipo de datos que tomaran las variables (En Python no es necesario declarar)
  - Nombres y valores de las constantes de un programa. (En principio en Python las simularemos poniendo un identificador en mayúsculas)
  - Implementación de las funciones que invocaremos en el cuerpo principal del programa

1.5.1. Bloque declarativo

1.5.1. Módulos o bibliotecas

- Una **biblioteca** es una colección de funciones que se encuentran recogidas en un archivo común, y que puede ser enlazada desde cualquier programa.
- La colección se construye con una cierta cohesión interna: hay librerías gráficas, matemáticas, estadísticas, lógicas, etc.
- Antes de programar nuevos algoritmos, es un buen consejo el de "bucear" en las librerías que acompañan

En Python: Un **módulo** (o **biblioteca**) es una colección de definiciones de variables, funciones, tipos,etc que pueden ser importadas para ser usadas desde cualquier programa.,

Ejemplo: from math import *

```
'''Esta llamada me permite calcular senos, cosenos, logaritmos, raíces
    cuadradas,  etc..'''
```

Información sobre el módulo: https://docs.python.org/2/library/math.html

1.5.2. Bloque declarativo

1.5.2. Datos y Tipos de Datos.

Datos: Son aquellos que un programa manipula. Sin datos un programa no funcionaría correctamente.

Tipo de dato: Es el conjunto de valores que puede tomar un dato durante el programa. Si se le intenta dar un valor fuera del conjunto se producirá un **error**.

Para afrontar este apartado se recomienda previamente la lectura de esta página:
https://sites.google.com/site/cursopython1/contenidos/modulo-2-sentencias

1.5. Bloque declarativo

1.5.2. Datos y Tipos de Datos. Clasificación

Simples	**numéricos**	**Enteros (int)**
		Reales (float)
		Complejos (complex)
	Lógicos (bool)	
	Nulo (NoneType)	
	Alfanuméricos(str)	

1.5. Bloque declarativo

1.5.2. Datos y Tipos de Datos. Clasificación

Compuestos

Vectores: Tema 3

Listas, Son la esencia de python, las podríamos definir como una secuencia ordenada de elementos encerrados entre corchetes y separados por comas.

l = [1, 4, 9, 16, 25]

Cadena de caracteres: **Tema 3**

Tuplas. Sería algo muy similar a listas, pero "inmutables". Esto quiere decir que, una vez creada una tupla, esta no se puede modificar.

t = 12345, 54321, 'hello!'

Diccionarios, tablas hash, tablas asociativas,... Tablas cuyo índice es una función a partir de un objeto, y el contenido será otro objeto.

Registros: Tema3

http://arsmentis.blogspot.com.es/2015/02/tipos-de-datos-compuestos-en-python.html

1.5. Bloque declarativo

1.5.2. Datos y Tipos de Datos. Curiosidades

En Python se nos enseña algo más que los datos y sus tipos:

- Determinados tipos son **inmutables** y otros **mutables (Conceptos que no tendremos en cuenta puesto que solamente utilizamos Python como instrumento para aprender Programación Estructurada)**.
- Los números, los strings y las tuplas son inmutables.
- Por el contrario, las listas y diccionarios son mutables.

http://www.mclibre.org/consultar/python/lecciones/python_variables_2.html

http://elclubdelautodidacta.es/wp/2012/09/python-el-sagrado-misterio-de-la-inmutabilidad/

1.5. Bloque declarativo

1.5.2.1. Tipos de datos Simples. Numéricos

Tipo entero:

En Python los enteros se representan con la palabra reservada int

Plataformas de 32bits: almacena números de -2.147.483.648 a 2.147.483.647, en plataformas de 64bits: 9.223.372.036.854.775.808 hasta

Para declarar un tipo entero o int es suficiente asignar un número entero a una variable,
ejemplo: entero = 23

Operaciones sobre datos de tipo entero:

+, -, *, División (/), Cociente de división (//), Resto de la división (%) y exponenciación(x**y = x^y) , además de las operaciones relacionales. Asignación aumentada a += b es igual que a= a+b,

1.5. Bloque declarativo

1.5.2.1 Tipos de datos Simples. Numéricos

Tipo real:

Subconjunto de los números reales.
Rango: ±2,2250738585072020 x 10-308 hasta ±1,7976931348623157×10308

Operaciones sobre datos de tipo real:

+, -, *, / y exponenciación, además de las operaciones relacionales.
En Python 3 el operador / siempre nos devuelve un resultado en punto flotante.

https://sites.google.com/site/cursopython1/contenidos/modulo-2-sentencias

1.5. Bloque declarativo

1.5.2.1. Tipos de datos Simples. Numéricos

Prioridad de los operadores:
Cuando hay más de un operador en una expresión, el orden de evaluación depende de las reglas de precedencia. Python sigue las mismas reglas de precedencia a las que estamos acostumbrados para sus operadores matemáticos:

1. **Paréntesis** tienen la precedencia más alta.

2. **La Exponenciación** tiene la siguiente precedencia más alta, así que 2**1+1 es 3 y no 4, y 3*1**3 es 3 y no 27.

3. **La Multiplicación y División** tienen las misma precedencia, que es más alta que la de la **Adición y la Subtracción**, que también tienen la misma precedencia. Así que 2*3-1 da 5 en lugar de 4, y 5-2*2 es 1, no 6.

4. **Los operadores con la misma precedencia** se evalúan de izquierda a derecha. En álgebra nos dicen que están asociados por la izquierda. Así, que en la expresión 6-3+2, la resta se hace primero, produciendo 3.

Lectura recomendada, prioridad operadores : http://gidsoft.org/operadores-aritmeticos-en-python/

1.5. Bloque declarativo

1.5.2.2. Tipos de datos Simples. Lógico

Tipo Lógico o Booleano: (bool)Sólo puede tomar dos valores: **True** y **False**. Las mayúsculas son importantes

```
>>> type(True) <type 'bool'>
```

Operadores lógicos: and, or, y not.

and	V	F
V	V	F
F	F	F

or	V	F
V	V	V
F	V	F

not	
V	F
F	V

1.5. Bloque declarativo

1.5.2.2. Tipos de datos Simples. Lógico

Operadores CONTINUACIÓN:

El operador == es uno de los **operadores de comparación**;
Los otros son:

```
>> x !=  y    # x no es igual a y
>> x  >  y    # x es mayor que y
>> x  <  y    # x es menor que y
>> x >= y    # x es mayor o igual que y
>> x <= y    # x es menor o igual que y
```

Prioridad de los operadores: not, and, or, <, >, ==, <=, >=, !=.

Ver pag. 45: Tabla completa de precedencia libro Introducción programación Python Andrés Marzal.

1.5. Bloque declarativo

1.5.2.3. Tipos de datos Simples. Carácter

Tipo Carácter: Conjunto finito de caracteres que la computadora reconoce. Este tipo sólo contiene un carácter, y suele ir entre comillas simples (depende del lenguaje). La mayoría de los ordenadores reconocen el siguiente juego de caracteres:

C. Alfabéticos: A..Z, a..z.
C. Numéricos: 0..9.
C. Especiales: +, -, ¡, (, $,.....

Las operaciones más habituales entre caracteres son las relacionales.

1.5. Bloque declarativo

1.5.2.3. Tipos de datos Simples. Cadena

Tipo cadena de caracteres: Sucesión de caracteres que se encuentran delimitados por comillas (simples o dobles, según el lenguaje de programación. Nosotros utilizaremos dobles). La longitud de una cadena es el número de caracteres que la componen (se encuentran dentro de los delimitadores “”).

Las operaciones básicas que se pueden realizar sobre cadenas son las relacionales. Mas adelante se verán otras funciones más avanzadas.

Ejemplo de tipo cadena de caracteres en Python:

```
>>  c = ‘Pepe’
>>  type (c)
>>  < class ‘str’>
```

1.5. Bloque declarativo

1.5.2.4. Tipos de datos Simples. None

Tipo None: Existe un valor especial en Python para representar aquellos casos en los que "no hay valor"; es el valor None.

Este valor es el único de su tipo.

Se interpreta como falso en las expresiones logicas y es el devuelto por las funciones que no devuelven ningún valor explícitamente.

Cuando el resultado de una expresion es None, el interprete no lo muestra:

```
>>> None
```

1.5. Bloque declarativo

1.5.3. Constantes

Constantes son datos que permanecen sin cambios durante todo el desarrollo del algoritmo (o ejecución del programa).

Propiedades:

1. No son necesarias para el correcto funcionamiento del programa. se puede utilizar directamente el valor, en vez de especificar el nombre que le dimos.

2. Su uso es adecuado para dar claridad, legibilidad al programa y para que resulte más fácil modificar el programa en un futuro .

Ejemplo: como declarar constantes

Algoritmo *ejemplo_1*
const
Nombre de la constante=valor;
b=0.5;
c="casa";

1.5. Bloque declarativo

1.5.3. Constantes

Constantes en Python:

En principio en Python no existen constantes, nosotros simularemos su existencia, creando un identificador que comience por una letra en mayúscula y asignándole un valor, el cual nunca modificaremos a lo largo de nuestro programa.

Ejemplo

```
# Archivo: creoConstante.py
# Autor: Alumno estrella
# Fecha: 1 del 10 de 2015
# Descripción: Ejemplo de creación de una constante con valor 3

Primera_constante=3
```

1.5. Bloque declarativo

1.5.4. Variables

Variables: Son datos cuyo valor puede cambiar durante el desarrollo del algoritmo. Internamente, en "**LENGUAJES ESTRUCTURADOS**" son posiciones de memoria que almacenan un valor que puede ser modificado durante la ejecución del módulo que las utiliza.

Propiedades en LENGUAJES ESTRUCTURADOS:

1. Al principio del algoritmo hay que indicar que variables se van a utilizar y qué tipo de datos van a almacenar (declaración de variables).

2. Las variables tendrán siempre un nombre que las identifica y un tipo de datos para identificar los valores que puede contener.

3. Declarar variables es solicitar al compilador que nos reserve una porción de memoria para ir almacenando un valor (que se modificará durante la ejecución si es necesario) del tipo de datos indicado.

1.5. Bloque declarativo

1.5.4. Variables

```
Algoritmo ejemplo_1
        const
                a=1;
                b=0.5;
                c="casa";
        VAR
                Lista de variables: tipo;
                 en1,en2:entero;
                re:real;
                car: carácter;
                cad1, cad2: cadena de caracteres;
```

1.5. Bloque declarativo

1.5.5. Variables en Python

- Una variable almacena un valor de cierto tipo.
- En Python no hace falta declarar explícitamente las variables antes de su uso, estas se crean en el momento en el que les asignamos un valor y se destruyen automáticamente al salir de su ámbito
- En Python podemos definir variables de tipo: Numérica (entera) Numérica (flotante) Numérica (complejo) Cadena de caracteres Booleano
- Para definir una variable en Python solo debemos definir un identificador y asignarle un valor

Ejemplo:
```
>>> a=3
>>> type(a)   <type 'int'>
>>> b=3.0
>>> type(b) <type 'float'>
>>> d='c'
>>> type(d) <type 'str'>
```

NOTA: Para distinguirlas de las constantes el identificador lo escribiremos en minúsculas

1.5. Bloque declarativo

1.5.4. Variables en Python

✓ Recordar:

1. Python no permite hacer referencia a una variable a la que nunca se le haya asignado un valor.

   >>> *print (b)*

2. El operador de asignación es el signo '=' no confundir con '=='.
3. El nombre de la variable es un identificador (nosotros lo escribiremos en minúscula)
4. Python es case_sensitive distingue entre mayúsculas y minúsculas
5. Dejaremos siempre un espacio a cada lado de:

   Asignación: a = a + 1

   Asignación aumentada (+=, *=, -=, etc): a += 1

   Comparaciones (==, <, >, !=, <=, >=, not, in, not in, etc): a == 3

   Operadores aritméticos: c = (a + b) * (c + d)

1.6. Bloque ejecutivo

Parte activa, claramente identificable con el contenido del algoritmo que se implementa a través del programa.

1.6. Bloque ejecutivo

1.6.1. Instrucciones de E/S.

1.6.2. Instrucciones de Asignación.

1.6.3. Instrucciones condicionales.

1.6.5. Bucles.

1.6. Bloque ejecutivo

Las sentencias ejecutables constituyen el resto del algoritmo. Deben ir entre las palabras reservadas Inicio y Fin:

Inicio
Sentencias
....
Fin

En Python:

No hay bloque ejecutivo, Pondremos un comentario
Cuerpo del programa

1.6. Bloque ejecutivo

1.6.1. Instrucciones de Entrada/Salida

Salida ó Escritura.

- de un programa son los datos que el programa proporciona al exterior :

```
print("Hola")
print('Hola')
```

- Las cadenas se pueden delimitar tanto por comillas dobles (") como por comillas simples (').

- La función print() admite varios argumentos seguidos. En el programa, los argumentos deben separarse por comas. Los argumentos se muestran en el mismo orden y en la misma línea, separados por espacios:

print(“hola", “amigos") salida: hola amigos

Profundizaremos en laboratorio, para ver los diferentes formatos de salida

1.6. Bloque ejecutivo

1.6.1. Instrucciones de Entrada/Salida

Entrada o Lectura. la "entrada" de un programa son los datos que llegan al programa desde el exterior (habitualmente teclado):

En Python: La función

input()

permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Intro, como muestra el siguiente ejemplo

```
>>print("¿Cuál es su nombre?")
>>nombre = input()
>>print("Encantada,", nombre)
```

Salida

```
>>>
        ¿Cuál es su nombre?
        Ana
        Encantada, ana
>>>
```

1.6. Bloque ejecutivo

1.6.1. Instrucciones de Entrada/Salida

De forma predeterminada, la función input() convierte la entrada en una cadena. Si se quiere que Python interprete la entrada como un número entero, se debe utilizar la función int() de la siguiente manera:

```
entero = int(input("Dígame un numero:"))
print(entero, “numero entero dado")
```

Entrada como un número decimal:

```
real = float(input("Dígame un numero:"))
print(real, “numero real dado")
```

1.6. Bloque ejecutivo

1.6.1. Instrucciones de Entrada/Salida

Ejemplo:

 # Archivo: XXXX_XX.py
 # Autor: XXXXX XXXXXXX (nombre y apellidos)
 # Fecha: DD/MM/AAAA
 # Descripción: Este programa es un ejemplo sencillo, uso de entrada/salida
#Constantes se declararían en esta parte A=1;
#Variables se declararían el tipo en esta parte e1,e2:entero; re:real; En Python no es necesario

```
A=1
```

#cuerpo del programa

```
ent1 = int(input("Introduzca primer numero: "))
ent2 = int(input(" Introduzca segundo numero: "))
print(A)
print("El valor de la variable ent1 es: ", ent1);
```

1.6. Bloque ejecutivo

1.6.1. Instrucciones de Entrada/Salida

Funcionamiento ejemplo

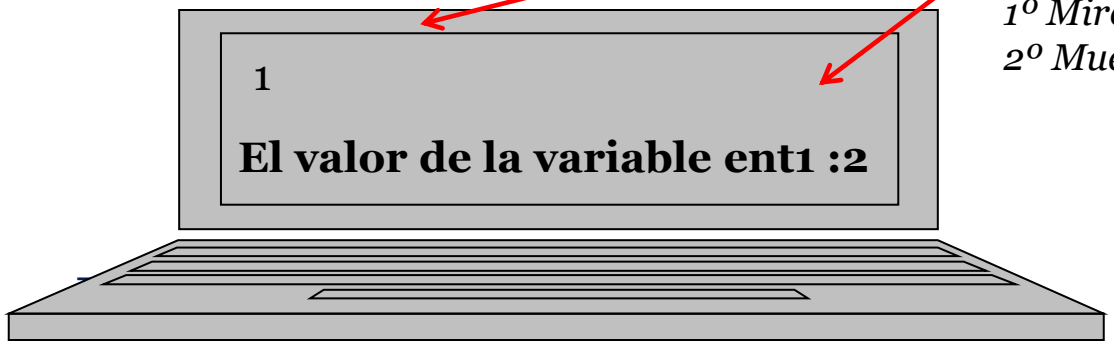
1º- Leer ent1, es de tipo entero, escribo en teclado 2

2º- Leer ent2, es de tipo entero, escribo en teclado 8

3º- Escribir A, En la pantalla de ejecución aparece un 1 ya que a es una constante con ese valor

Variable que identifica dirección	Memoria RAM
ent1	2
ent2	8
re	

3º- Escribir en pantalla parte textual entre comillas *el valor de la variable ent1 es el valor que tiene ent1 en memoria RAM:*
1º Miro valor en RAM
2º Muestro en pantalla.



1.6. Bloque ejecutivo

1.6.2. Instrucciones de Asignación

La asignación permite dar a una variable un valor concreto.
Sintaxis de una asignación:

nombreVariable = Expresión* | ***nombreVariable*** | ***NombreConstante***

A- Si la parte derecha del operador de asignación es una expresión: el valor devuelto por esta debe ser del mismo tipo que el de la variable.

B- Si en la parte derecha aparece otro nombre de variable, se asignará su valor, y por lo tanto ambas deben de ser del mismo tipo.

C- Si el valor a asignar es el de una constante, también debe coincidir con el tipo de la variable de la parte izquierda.

El símbolo que representa la operación en Python es =.

* una **expresión** es una combinación de constantes, variables o funciones, que es interpretada de acuerdo a las normas particulares de precedencia y asociación para un lenguaje de programación en particular

1.6. Bloque ejecutivo

1.6.2. Instrucciones de Asignación EJEMPLO PSEUDOCÓDIGO

Ejemplo:

```
Algoritmo ejemplo_2;
const
                    A=1;
                    Pi=3.14;
VAR
                     ent1,ent2:entero;
                    re:real;
 INICIO
                    ent1 = 2+A;
                    Escribir(" valor expresión",ent1);
                     ent2 = ent1;
                    Escribir ("el valor de la variable ent1 en ent 2: ", ent2);
                    re = Pi;
                    Escribir ("el valor de la constante ", re);

FIN
```

1.6. Bloque ejecutivo

1.6.2. Instrucciones de Asignación

La asignación en python:

En python no hace falta declarar explícitamente las variables antes de su uso, estas se crean en el momento en el que les asignamos un valor y se destruyen automáticamente al salir de su ámbito (El ámbito de las variables indica el espacio en el que la variable es visible o dicho de otra forma, si se puede utilizar).

Al asignarle un valor a una variable, esta automáticamente coge el valor y el tipo del dato que le hayamos pasado. Esto es lo que se conoce como "***tipado dinamico" no lo utiliremos en esta asignatura, ya que los lenguajes estructurados no tienen este tipo de tipado***

```
>>> a=3
>>> type(a) <type 'int'>
>>> a=3.0
>>> type(a) <type 'float'>
>>> a='c'
>>> type(a) <type 'str'>
```

Después de la última asignación de la variable *a*, esta es del tipo *carácter*

1.6. Bloque ejecutivo

1.6.2. Instrucciones de Asignación

La asignación en python:

El nombre de una variable, es un identificador.
Python nos permite asignaciones múltiples.

```
>>> ciudad = municipio = 'Tarragona'
>>> ciudad
        'Tarragona'
>>> municipio
        'Tarragona'
```

Recomendaciones:
Dejaremos siempre un espacio a cada lado de:

- Asignación: a = a + 1
- Asignación aumentada (+=, *=, -=, etc): a += 1
- Comparaciones (==, <, >, !=, <=, >=, not, in, not in, etc): a == 3
- Operadores aritméticos: c = (a + b) * (c + d)

1.6. Bloque ejecutivo

1.6.2. Instrucciones de Asignación. En Python

Ejemplo:

 # Archivo: XXXX_XX.py
 # Autor: XXXXX XXXXXXX (nombre y apellidos)
 # Fecha: DD/MM/AAAA
 # Descripción: Este programa es un ejemplo sencillo asignaciones

```
A = 1
Pi = 3,14
# Inicio
ent1 = 2+A
print("valor expresión ",ent1)
ent2 = ent1
print ("el valor de la variable ent1 en ent 2: ", ent2)
re = Pi
print ("el valor de la constante ", re)
# Fin
```

1.6. Bloque ejecutivo

1.6.2. Instrucciones de Asignación

Ejemplo:

 # Archivo: XXXX_XX.py
 # Autor: XXXXX XXXXXXX (nombre y apellidos)
 # Fecha: DD/MM/AAAA
 # Descripción: Repetición ejemplo anterior utilizando valor de constante pi contenida en biblioteca math
 # *Inicio*

```
from math import * # Desde aquí obtengo constante pi
A = 1
ent1 = 2+A
print("valor expresión ",ent1)
ent2 = ent1
print ("el valor de la variable ent1 en ent 2: ", ent2)
re = pi
print ("el valor de la constante ", re)
```

 # Fin

1.6. Bloque ejecutivo

1.6.2. Instrucciones de Asignación

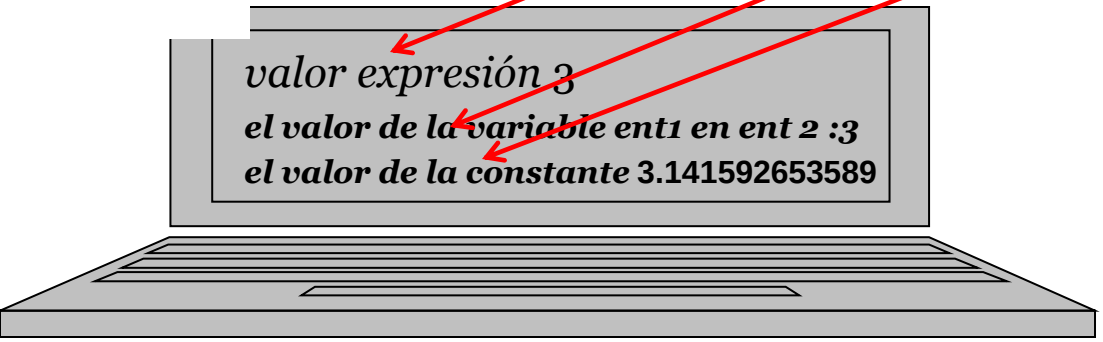
Funcionamiento ejemplo

1º- Asigno a ent1 =2 + valor constante a (1) = 3
Escribo ese valor en pantalla

2º- Asigno a ent2 el valor que tengo en ent1
y escribo mensaje en pantalla

3º- Asigno a variable valor de constante y la escribo

Variable que identifica dirección	Memoria RAM
ent1	3
ent2	3
re	3.14



EJEMPLOS ILUSTRATIVOS

INSTRUCCIONES DE ASIGNACIÓN Y DE E/S

Ejemplos ilustrativos

1. Realizar un algoritmo para intercambiar los valores de 2 variables.

Solución:

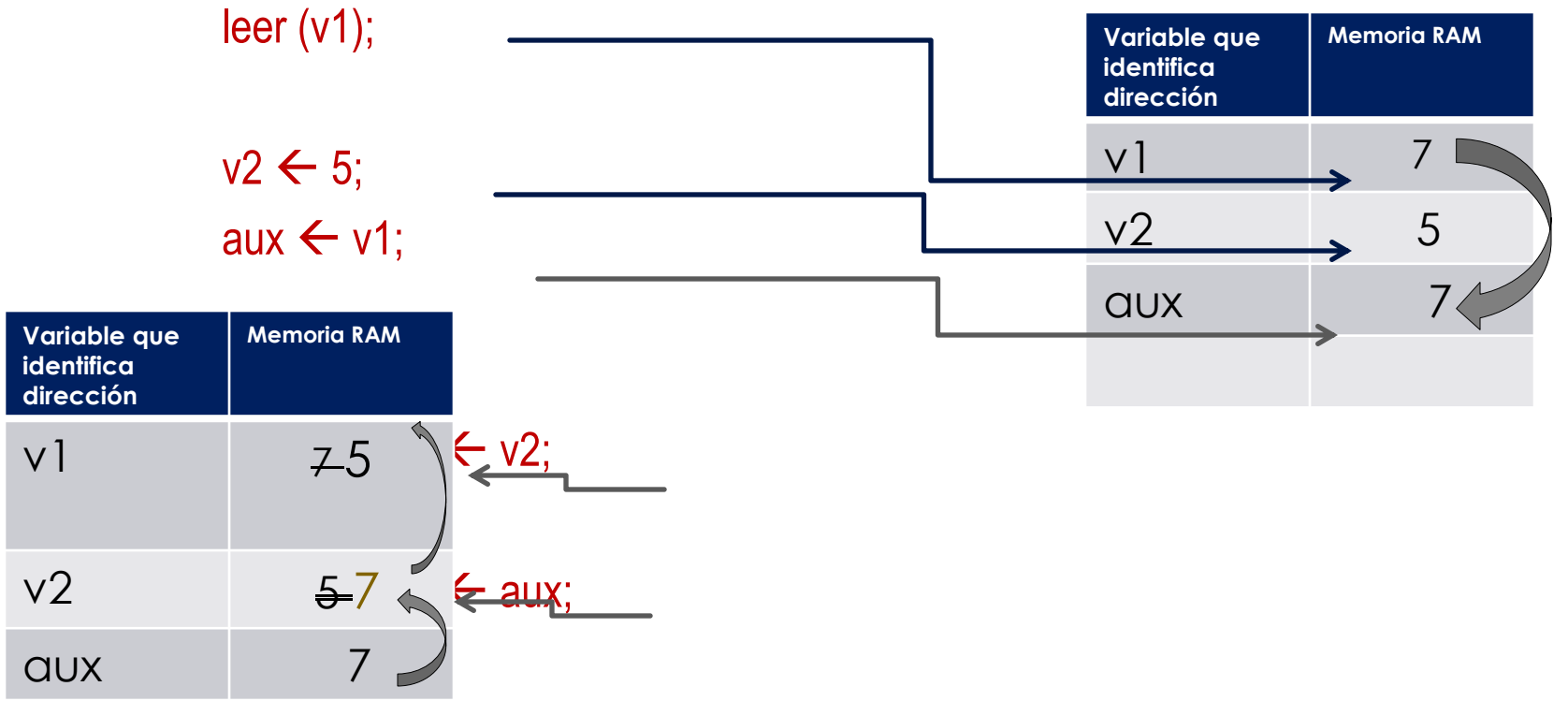
 # Archivo: XXXX_XX.py
 # Autor: XXXXX XXXXXXX (nombre y apellidos)
 # Fecha: DD/MM/AAAA
 # Descripción: Intercambio de dos valores
 # Inicio

```
v1 = int(input('Introduce primer entero :'))
v2 = 5
aux = v1
v1 = v2
v2 = aux
```

 # Fin

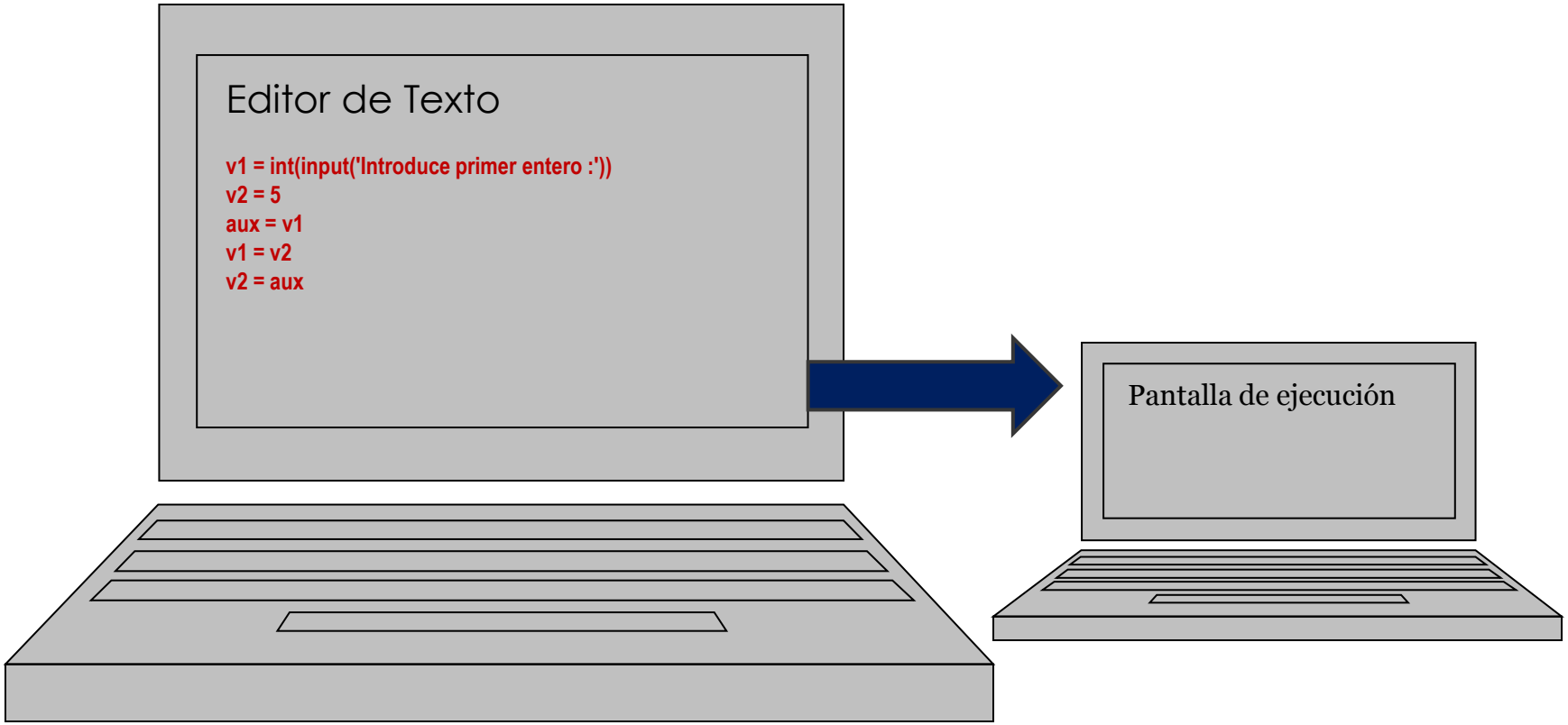
Ejemplos ilustrativos

Interpretación de la solución:



Ejemplos ilustrativos

Interpretación de la solución:

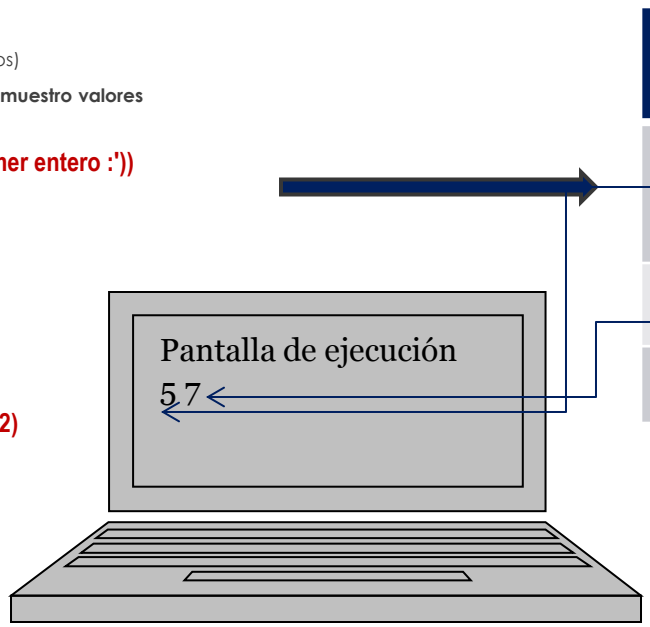


Ejemplos ilustrativos

Interpretación de la solución: Añado al algoritmo línea para mostrar valor de variables

Algoritmo Intercambio;

```
# Archivo: XXXX_XX.py
# Autor: XXXXX XXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Intercambio de dos valores muestro valores
# Inicio

    v1 = int(input('Introduce primer entero :'))

    v2 = 5

    aux = v1

    v1 = v2

    v2 = aux

    print(' valores v1 y v2 ', v1, v2)


fin
```



Variable que identifica dirección	Memoria RAM
v1	~~7~~ 5
v2	~~5~~ 7
aux	7

Ejemplos ilustrativos

2. Calcular el área de un cuadrado :

1º Asignando el valor 3 a la longitud del lado.

2º Asignando un valor a la longitud del lado desde el teclado

Solución 1:**Asignando el valor 3 a la longitud del lado.**

```
# Archivo: XXXX_XX.py
# Autor: XXXXX XXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Area cuadrado asignando lado 3 "SOLUCIÓN1"
# Inicio

    longitud = 3
    area = longitud*longitud
    print(' el area del cuadrado de base 3 es ', area)

# Fin
```

Ejemplos ilustrativos

Solución 2: **Asignando un valor a la longitud del lado desde el teclado.**

```
# Archivo: XXXX_XX.py
# Autor: XXXXX XXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Asignando un valor a la longitud del lado desde el teclado
# Inicio
    longitud = int(input(' Dime el valor del lado: '))
    area = longitud**2 # Otra forma para el cálculo del  cuadrado de un número
    print(' el area del cuadrado de lado ', longitud,' es ', area)
# Fin
```

Ejemplos ilustrativos

En la solución de este ejemplo puede surgir el ***problema*** de introducir por teclado longitudes de lados negativos ó cero, con lo cual el ejercicio sería incorrecto.

Solución : Uso de instrucciones condicionales (ver apartado 1.6.3.).

TRABAJO PERSONAL

Trabajo personal

Lectura recomendada

Cap 3 Libro Introducción Programación Python (Andrés Marzal).

Laboratorio: Hoja Tema 1 parte 1.

Resolver ejemplos con Python empleando la sentencias más adecuada:

1. Suma de 2 números.
2. Calculo de la longitud de una circunferencia.
3. Calculo del cuadrado de un número.
4. *Calculo de la longitud y área de círculo para un radio dado.*

1.6. Bloque ejecutivo

- 1.6.1. Instrucciones de E/S.
- 1.6.2. Instrucciones de Asignación.
- 1.6.3. Instrucciones condicionales.
- 1.6.5. Bucles.
- 1.6.5. Excepciones

1.6. Bloque ejecutivo

1.6.3. Instrucciones Condicionales

La instrucción condicional básica permite elegir entre dos opciones, dependiendo del valor de una expresión lógica. La sintaxis completa de esta instrucción:

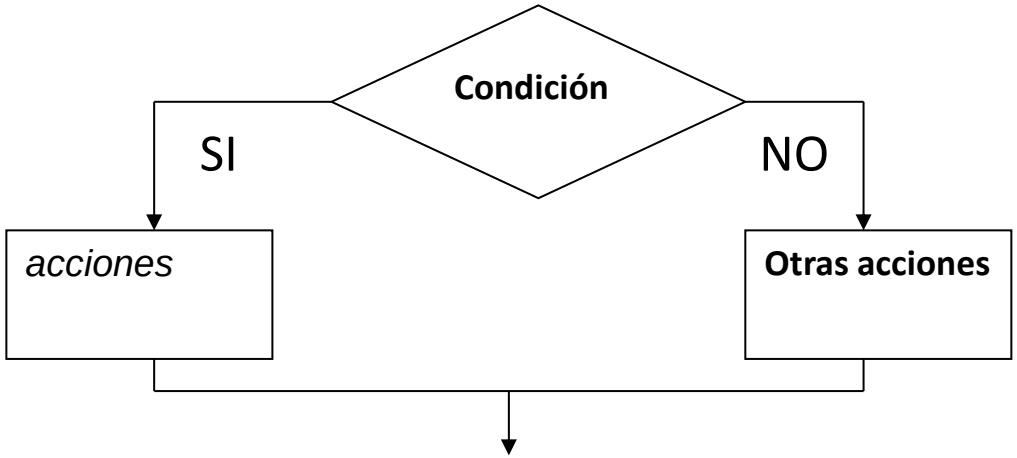
```
if condición:
        instrucciones
else:
        instrucciones
```

condición es una expresión lógica (devuelve verdadero o falso). Si el resultado de evaluar la expresión es verdadero, se ejecutarán las instrucciones que siguen al "entonces"; si es falso, se ejecutarán las instrucciones que siguen al "si_no".

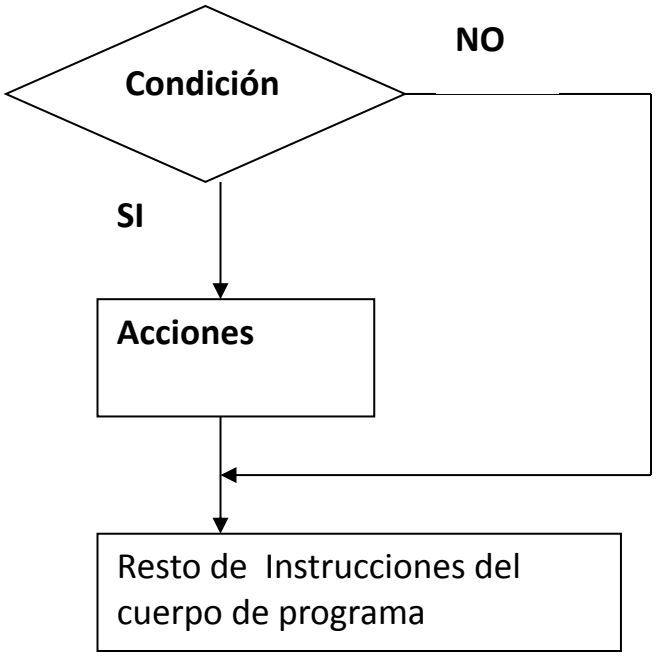
1.6. Bloque ejecutivo

1.6.3. Instrucciones Condicionales

Diagrama de flujo: if condición:
acciones
else:
otras acciones



if condición:
acciones



1.6. Bloque ejecutivo

1.6.3. Instrucciones Condicionales

Concepto de indentación en PYTHON:

Una de las característica principales de este lenguaje es **que los bloques se definen por su indentación**.
En las instrucciones condicionales las sentencias que se ejecutan dentro de cada bloque vienen marcadas por la indentación

Se recomienda el uso de espacios frente a tabs.

```
>>> if condicion:
...                 Bloque principal de ejecución()
...     Bloque fuera de instrucción if()
>>>
```

EJEMPLOS ILUSTRATIVOS

INSTRUCCIÓN CONDICIONAL

Ejemplos ilustrativos

1.6.3. Instrucciones Condicionales

Ejemplo 1:
Recordemos el ejemplo ilustrativo del apartado 1.6.2.

2. Calcular el área de un cuadrado a partir de la longitud de uno de sus lados. Volverlo a implementar añadiendo que la longitud del lado ha de ser positiva.

```
longitud = int(input(' Dime el valor del lado: '))
if longitud > 0:
        area = longitud**2
        print(' el área del cuadrado de lado ', longitud,' es ', area)
else:
        print('el valor del lado es incorrecto')
```

NOTA: Obsérvese la indentacción de instrucciones

Ejemplos ilustrativos (Instrucciones condicionales)

1.6.3. Instrucciones Condicionales

Ejemplo2:
Calcular el logaritmo neperiano de un número, teniendo en cuenta que si el número es negativo se calcula el logaritmo del valor absoluto del número.

```
# Programa Logaritmo neperiano
from math import *
n = int(input('Introduzca un número:  '))
if n<0:
        print ('No se puede calcular el logaritmo de un número negativo,
                tomaremos el valor absoluto de ', n)
        n = abs(n)
logaritmo = log(n)
print ('El logaritmo neperiano de ', n, 'es: ', logaritmo)
```

Ejemplos ilustrativos (Instrucciones condicionales)

1.6.3. Instrucciones Condicionales

Ejemplo3:

Dado un número entre 1 y 7 mostrar el día de la semana con el que se corresponde

```
# Programa dia de la semana
n = int(input('introduzca el día de la semana '))
if n<0 or n>7:
    print ('dia incorrecto ')
else:
    if n == 1:
        print('Lunes')
    else:
        if n == 2:
            print('Martes')
        else:
            if n == 3:
                print('Miércoles')
```

Ejemplos ilustrativos (Instrucciones condicionales)

1.6.3. Instrucciones Condicionales

```
else:
        if n == 4:
            print('Jueves')
        else:
            if n == 5:
                print('Viernes')
            else:
                if n == 6:
                    print('Sábado')
                else:
                    if n == 7:
                        print('Domingo')
```

http://www.mclibre.org/consultar/python/lecciones/python_if_else.html

1.6. Bloque ejecutivo

1.6.3. Instrucciones Condicionales

Cuando se trata de casos múltiples y excluyentes unos a otros (como en este último ejemplo) en la mayoría de los lenguajes estructurados se dispone de otra sentencia que permite especificar múltiples alternativas, se trata de la instrucción ***segun***, cuya sintaxis es:

```
segun expresión hacer
        cte | cte, cte, ...: instrucciones
        cte | cte, cte,....:  instrucciones
        ......
        [si_no]: instrucciones
fin según
```

El tipo del resultado de evaluar la expresión debe ser el mismo que el de las constantes,
En C sentencia switch ...case

Ejemplos ilustrativos

1.6.3. Instrucciones Condicionales

Ejemplo 3 con instrucción según:

```
Algoritmo dia_semana_bis;
Variable   dia: entero
Inicio
        Leer (dia)
        Segun dia hacer
                1: Escribir ("Lunes")
                2: Escribir ("Martes")
                3: Escribir ("Miércoles")
                4: Escribir ("Jueves")
                5: Escribir ("Viernes")
                6: Escribir ("Sábado")
                7: Escribir ("Domingo")
                si_no: Escribir ("El número introducido no es válido")
        fin segun
Fin
```

http://edupython.blogspot.com.es/2014/07/donde-quedo-el-switch-case.html

Ejemplos ilustrativos

1.6.3. Instrucciones Condicionales

En PYTHON:

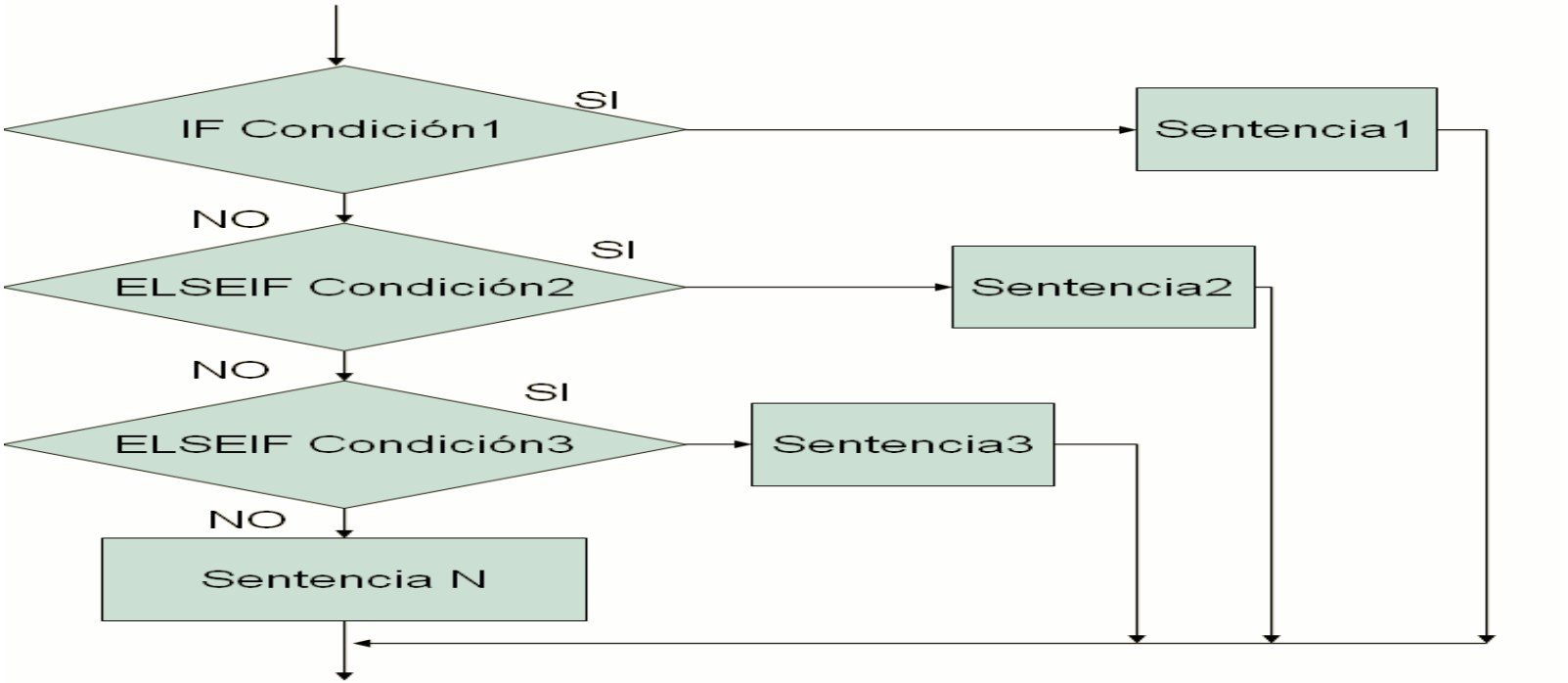
Carece de esta sentencia. Para simular esta instrucción podemos utilizar la secuencia de instrucciones:

if-elif-else

La instrucción **elif** es una contracción de las instrucciones **else** e **if**, con la ventaja de que no es necesario añadir un nivel de indentación al momento de utilizarla. Si no existiera la instrucción **elif**, el código de arriba se tendría que escribir de una forma un tanto menos conveniente

Ejemplos ilustrativos (Instrucciones condicionales)

1.6.3. Instrucciones Condicionales



Ejemplos ilustrativos (Instrucciones condicionales)

1.6.3. Instrucciones Condicionales

Ejemplo 3 con instrucción elif:

```
# Programa día de la semana
n = int(input('introduzca el día de la semana '))
if n<0 or n>7:
    print ('dia incorrecto ')
elif n == 1:
    print('LUNES')
elif n == 2:
    print('MARTES')
elif n == 3:
    print('MIÉRCOLES')
elif n == 4:
    print('JUEVES')
elif n == 5:
    print('VIERNES')
elif n == 6:
    print('SÁBADO')
else:
    print('DOMINGO')
```

http://edupython.blogspot.com.es/2014/07/donde-quedo-el-switch-case.html

TRABAJO PERSONAL

Trabajo personal

Lectura recomendada

Cap 4, Apartado 4.1: Libro Introducción Programación Python (Andrés Marzal).

Laboratorio: Hoja Tema 1 parte 2.

Resolver ejemplos con Python empleando la instrucción condicional más adecuada:

1. Dado un dato entero ver si es positivo.
2. Dada una nota numérica obtener su calificación nominal.
3. Dado el lado de un cubo, mostrar su área y su volumen, teniendo en cuenta que el lado sea mayor que 0.
4. *Obtener las soluciones reales o complejas de una ecuación de 2º grado ax^2+bx+c=0 dados a, b y c.*
5. Desarrollar un algoritmo que tras leer 3 enteros los escriba por pantalla ordenados.

1.6. Bloque ejecutivo

- 1.6.1. Instrucciones de E/S.
- 1.6.2. Instrucciones de Asignación.
- 1.6.3. Instrucciones condicionales.
- 1.6.4. Excepciones
- 1.6.5. Bucles

1.6. Bloque ejecutivo

1.6.4. Excepciones

Las excepciones son errores detectados por Python durante la ejecución del programa. Cuando el intérprete se encuentra con una situación excepcional, como el intentar dividir un número entre 0 o el intentar acceder a un archivo que no existe, este genera o lanza una excepción, informando al usuario de que existe algún problema.

En Python, contamos con 3 palabras reservadas (y una palabra opcional) para el manejo de excepciones:

try: Dentro del bloque try se ubica todo el código que pueda llegar a *levantar* una excepción, se utiliza el término *levantar* para referirse a la acción de generar una excepción.

except: Se encarga de capturar la excepción y nos da la oportunidad de procesarla mostrando por ejemplo un mensaje adecuado al usuario

finally : Finalmente, puede ubicarse este bloque, donde se escriben las sentencias de finalización, que son típicamente acciones de limpieza. La particularidad del bloque **finally** es que se ejecuta siempre, haya surgido una excepción o no. Si hay un bloque except, no es necesario que esté presente el finally, y es posible tener un bloque **try** sólo con **finally**, sin **except**.

1.6. Bloque ejecutivo

1.6.4. Excepciones

```
#Constantes
Año_actual = 2015

#Cuerpo del programa
x = input("Introduzca el año en que nacio: ")
try:
    año_nacimiento = int(x)
except: print("Algo ha fallado: no es posible calcular su edad")

edad = Año_actual - año_nacimiento
print("Su edad es: ", edad , " años")
```

```
>>>
Introduzca el año en que nacio: ASD
Algo ha fallado: no es posible calcular su edad
Traceback (most recent call last):
  File "C:/Windows/System32/EX.py", line 8, in <module>
    edad = Año_actual - año_nacimiento
NameError: name 'año_nacimiento' is not defined
>>>
Introduzca el año en que nacio: 1998
Su edad es:  17  años
>>>
```

1.6. Bloque ejecutivo

1.6.4. Excepciones

```
dividendo = 5
divisor = 0
try:
   cociente = dividendo / divisor
except:
   print ('No se permite la división por cero')
```

```
>>>
>>>
No se permite la división por cero
>>> >>>
```

1.6. Bloque ejecutivo

- 1.6.1. Instrucciones de E/S.
- 1.6.2. Instrucciones de Asignación.
- 1.6.3. Instrucciones condicionales.
- 1.6.4. Excepciones
- 1.6.5. Bucles

1.6. Bloque ejecutivo

1.6.5. Bucles

Un bucle engloba una secuencia de *acciones que se describen una sola vez*, pero que *se pueden ejecutar varias veces*.

Las acciones englobadas se llaman ***CUERPO del*** bucle y cada ejecución del cuerpo se denomina ***ITERACIÓN.***

Manejaremos tres tipos de bucles en programación estructurada:

Bucle ***desde***
Bucle ***mientras***
Bucle ***repetir***

1.6. Bloque ejecutivo

1.6.5. Bucles

Diferencia fundamental de estos 3 tipos de estructuras de control repetitivas:

"***Es que permiten ejecutar las acciones un número de veces que puede estar definido a priori o indefinido hasta que se cumpla una determinada condición booleana***."

1.6. Bloque ejecutivo

1.6.5.1. Bucle For

Los bucles FOR se utilizan cuando queremos que se repita una serie de instrucciones un número determinado de veces
Sintaxis más habitual :

```
for contador in elemento recorrible (lista, cadena, range, etc.):
        cuerpo del bucle (bloque acciones indentadas)
```

- No es necesario definir la variable de control antes del bucle.
- Se puede utilizar como variable de control una variable ya definida en el programa.
- El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento recorrible (elementos de una lista o de un range(), caracteres de una cadena, etc.).

NOTA: range() tipo de dato subrango se estudiará en profundidad Tema3

1.6. Bloque ejecutivo

1.6.5.1. Bucle For

Cuando se programa un bucle FOR, se escribe una primera instrucción con la que se dan los tres parámetros fundamentales del bucle FOR.

1º **El VALOR INICIAL** de la variable del bucle. Este será el valor que tendrá la variable la primera vez que ejecutan las instrucciones que se encuentran dentro del bucle.

2º **El VALOR FINAL** de la variable del bucle. Mientras que la variable del bucle FOR sea MENOR O IGUAL al valor final, se ejecutarán las instrucciones que se encuentran dentro del bucle.

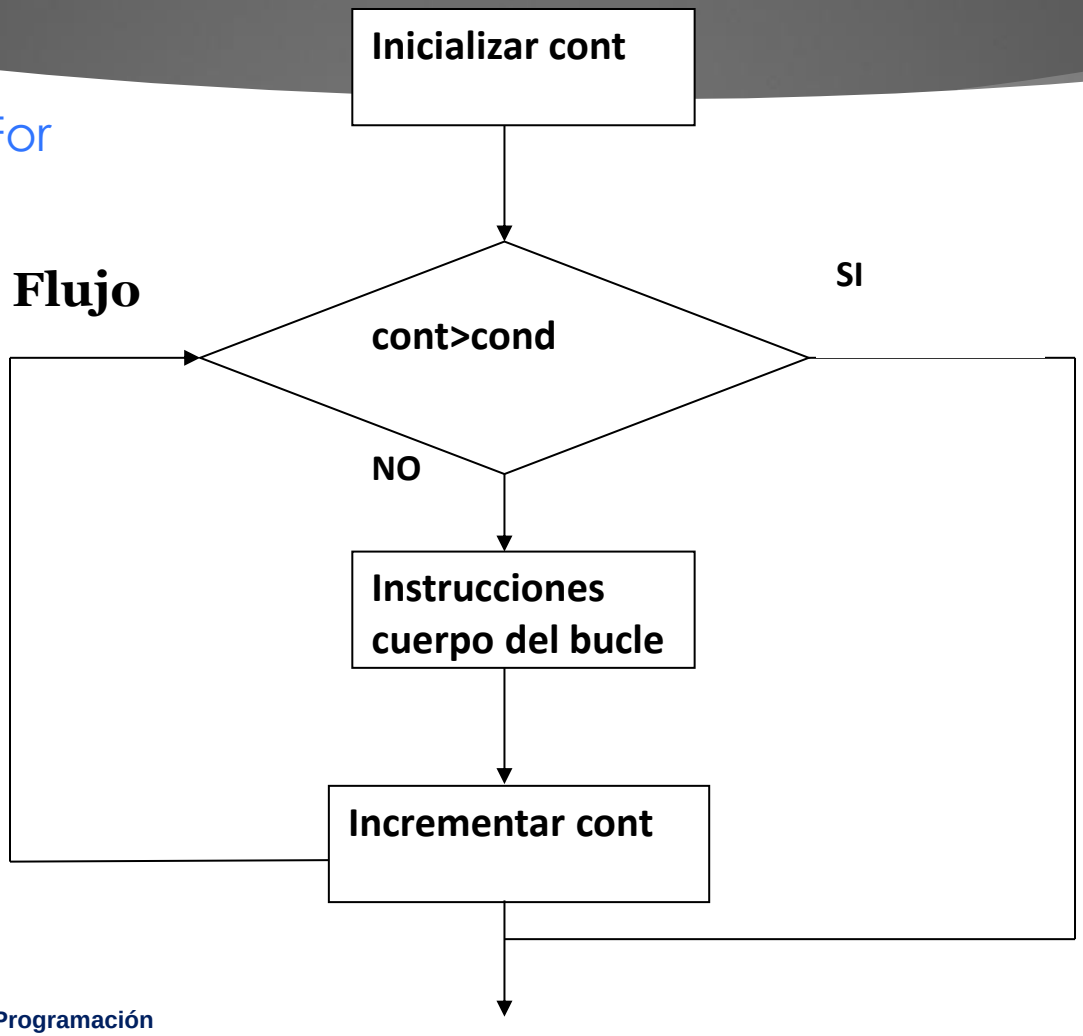
3º **El PASO** del bucle. Esta es una herramienta fundamental dentro del bucle FOR y que hace que sea muy diferente a un bucle WHILE. Y es que **la variable del bucle MODIFICA SU VALOR AUTOMÁTICAMENTE POR CADA PASO DEL BUCLE**.

¿Cómo lo modifica? valiéndose del PASO, de forma que, por cada vez que se ejecutan las instrucciones que se encuentran dentro del bucle FOR, se incrementa AUTOMÁTICAMENTE la variable del bucle FOR un valor igual al PASO.

1.6. Bloque ejecutivo

1.6.5.1. Bucle For

Diagrama de Flujo



Ejemplo ilustrativo

1.6.5.1. Bucle For

El tipo range()
con un único argumento se escribe range(n) y crea una lista creciente de n términos enteros que empieza en 0 y acaba antes de llegar a n (los términos aumentan de uno en uno). Es decir,

range(n) = range(0, n) = [0, 1, ... , n-1].
range(m, n) = [m, m+1, ... , n-1]
range(m, n, p) = [m, m+p, ... , x] siendo x tal que n - p ≤ x < n (si p es negativo, n - p ≥ x > n)

Ejemplo 1: Escribir sentencias un número de veces.

```
for i in range(5):
        print ("No volveré a mascar chicle en clase.")
```

Salida: No volveré a mascar chicle en clase.
No volveré a mascar chicle en clase.
No volveré a mascar chicle en clase.
No volveré a mascar chicle en clase.
No volveré a mascar chicle en clase.

Más información tipo range: http://www.mclibre.org/consultar/python/lecciones/python_range.html#L182

Ejemplo ilustrativo

1.6.5.1. Bucle For

Ejemplo 2: **Calcular el factorial de un número.**

```
# Algoritmo factorial
fact = 1
n= int(input('Introduzca el número para calcular el factorial :'))
if n != 0:
   for i in range(2,n+1):
      fact = fact*i
print(fact)
```

Ejemplo ilustrativo (bucle For)

1.6.5.1. Bucle For

Ejemplo 3: **Escribir las tablas de multiplicar**

```
#Programa tablas_multiplicar;
for i in range(1,11):
    print('Tabla del :', i)
    for j in range(1,11):
      prod = i*j
      print( i,'*', j, '=', i*j)
```

Nota: *Para desarrollar el algoritmo hemos utilizado bucles anidados.*

Ejemplo ilustrativo

1.6.5.1. Bucle For

Ejemplo 4: **Imprimir números pares del 2 al 10**

```
# Dos formas de imprimir los números pares del 2 al 10
#1ª FORMA

for i in range(2,12,2):
        print(i)

#2ª FORMA

for i in range(5):
        print ((i + 1) * 2)
```

1.6. Bloque ejecutivo

1.6.5.2. Bucle while

Un bucle **while** se usa cuando el programa debe iterarse hasta que se alcanza una determinada condición.

Sintaxis :

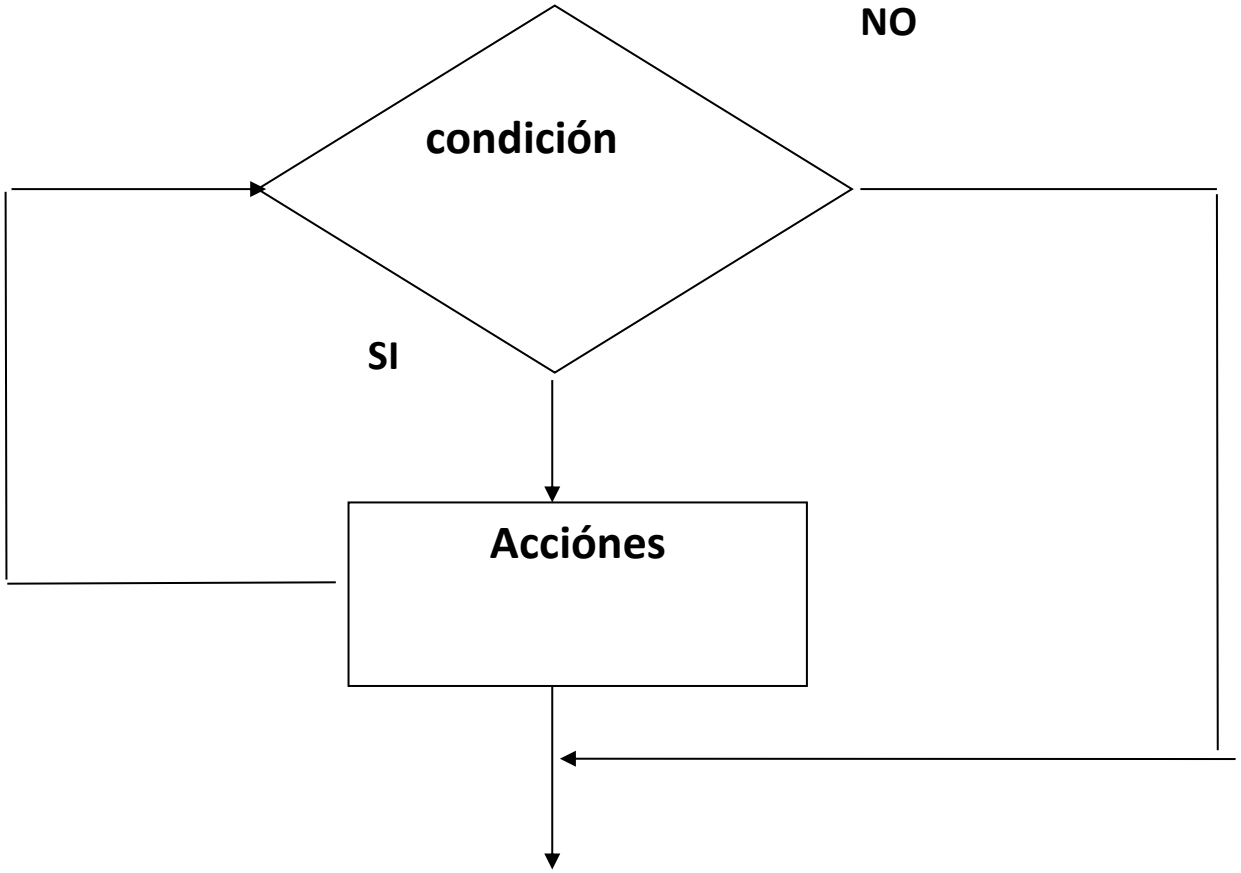
```
while condición:
        acciones (indentadas)
```

En esta estructura las instrucciones sólo se ejecutarán si se cumple una determinada **condición**.
Hay que estudiar detenidamente la condición (expresión lógica) para que el bucle termine, esto significa que se debe de ir modificando el valor de elementos de la condición en las instrucciones a realizar para llegar a la condición de parada.

1.6. Bloque ejecutivo

1.6.5.2. Bucle while

Diagrama de Flujo



1.6. Bloque ejecutivo

1.6.5.2. Bucle while

Técnicas para control de bucles en programación Estructurada:

1. **Utilización de un valor centinela**, es decir un valor especial utilizado para indicar el final de una lista de datos y que no se procesa como dato válido. Vista en ejemplo anterior.

2. **Utilización de una bandera o interruptor**, es decir una variable (generalmente lógica) que indica el estado: si es verdadero, se sigue iterando. Por ejemplo, parar cuando el usuario introduzca un número impar:

1.6. Bloque ejecutivo

1.6.5.2. Bucle while

Técnicas para control de bucles:

1. Utilización de un valor centinela, es decir un valor especial utilizado para indicar el final de una lista de datos y que no se procesa como dato válido. Ver ejemplo ilustrativo1.

1.6. Ejemplo ilustrativo

1.6.5.2. Bucle while

Ejemplo1: Calcular la media aritmética de una serie de números que se introducen por teclado hasta que el usuario introduzca -1.

```
#Programa media aritmética;
cont = 0
suma = 0
n = int(input('Introduce un número positivo (-1 para terminar): '))
while n != -1: #CENTINELA
    suma = suma + n
    cont = cont + 1
    n = int(input('Introduce un número positivo (-1 para terminar): '))
media = suma /cont
print ('La media aritméticas es: ', media)
```

Ejemplo ilustrativo

1.6.5.2. Bucle while

2. Empleo de bandera : es decir una variable (generalmente lógica) que indica el estado: si es verdadero, se sigue iterando. Por ejemplo, parar cuando el usuario introduzca un número impar:

Ejemplo2 :

```
seguir = True
while seguir:
        n=int(input('leer un número: '))
        if n % 2 != 0:
            seguir = False
print(' Cuando salga del bucle el número ',n,' es impar ')
```

Ejemplo ilustrativo

1.6.5.2. Bucle while

Ejemplo3 : Obtener si un número es primo o no

```
#Programas número_primo;

n = int(input('Lee el numero '))
aux = 2
primo = True
while (aux <n) and (primo):
    #La variable "primo" se considera un interruptor (switch) o bandera (flag)
    if n % aux != 0:
        aux = aux + 1
    else:
        primo = False
#cuando hay dos condiciones de salida suele ser necesario averiguar por cual de las dos se ha salido
if primo:
    print('El número es primo')
else:
    print ('El número no es primo')
```

1.6. Bloque ejecutivo

1.6.5.3. Bucle Repetir

Una de las estructuras de control de ejecución que no está presente en el lenguaje de programación Python es el bucle REPETIR.

Este bucle aparece en la mayoría de los lenguajes de programación (do while en lenguajes como C ó JAVA)

La diferencia de este bucle con el bucle while es que en este bucle las instrucciones se ejecutan al menos una vez, puesto que la condición de parada se evalúa al final, después de la primera ejecución;

primero se ejecutan las instrucciones y después se evalúa la condición, si es falsa, se vuelve a repetir el proceso y si no, si es cierta, finaliza el bucle es decir se sale del bucle.

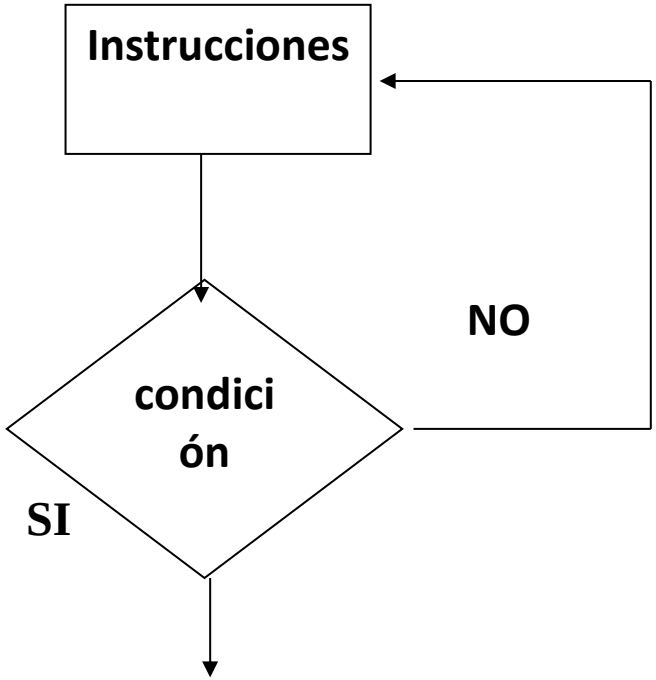
1.6. Bloque ejecutivo

1.6.5.3. Bucle Repetir

Sintaxis :

Repetir
instrucciones
hasta *condición*

Diagrama de Flujo



1.6. Bloque ejecutivo

1.6.5.3. Bucle Repetir

Diferencias entre bucle *repetir* y bucle *while*:

1ª- la condición se evalúa después de ejecutar el cuerpo del bucle, al menos éste se realiza una vez aunque la condición sea siempre verdadera.

2ª- La segunda, es que para que se siga ejecutando, la condición ha de ser falsa y no verdadera.

También en este bucle la expresión lógica habrá de ser variable es decir se deberá modificar dentro del cuerpo del bucle, para evitar que sea un bucle infinito.

1.6. Bloque ejecutivo

1.6.5.3. Bucle Repetir

IMPLEMENTACIÓN DEL BUCLE REPETIR EN PYTHON

1. En Python no existe este bucle.
2. Para los algoritmos que lo requieran en Python, implementaremos bucle "infinito" (usando un instrucción while True:) y añadiremos una instrucción if con un break al final del cuerpo

#En este ejemplo siempre leeremos una cadena, el programa para cuando escriba si

```
# 1ª forma de simular repeat
while True:
    entrada = input("¿Quieres salir? ")
    if entrada == "si":
        break
# 2ª forma de simular repeat
entrada = "no"
while entrada!= "si":
    entrada = input("¿Quieres salir? ")
```

1.6. Bloque ejecutivo

1.6.5.3. Bucle Repetir

NOTA IMPORTANTE:

EN PROGRAMACIÓN ESTRUCTURADA ESTÁ PROHIBIDO UTILIZAR BREAK.

Nosotros solamente podremos utilizar este procedimiento para simular el bucle repetir ya que Python no tiene esta instrucción, que es muy importante a la hora de utilizar un lenguaje estructurado.

Ejemplo ilustrativo

1.6.5.3. Bucle Repetir

Ejemplo1 : Encontrar el mayor número positivo N tal que la suma de 1+2+....+N sea estrictamente menor que un valor positivo límite dado. (IMPLEMENTACIÓN EN PYTHON)

```
#Programa Mayor_positivo;
while True:
    lim= int(input('Introduzca el valor del límite '))
    if lim >0:
        break
num = 0
suma= 0
while True:
    num += 1;#num=num+1
    suma += num; #suma=suma+ num
    if suma >= lim:
            break
suma = suma - num;
num = num - 1;
print('La suma desde 0 hasta ', num, ' = ', suma,‘ está  inmediatamente por debajo de ', lim);
```

Ejemplo ilustrativo (Bucle repetir)

1.6.5.3. Bucle Repetir

Ejemplo2: Realizar la media de 10 números impares que introduzca el usuario.

```
#Programa Media_impar;
suma = 0
cont = 0
while True:
    while True:
        n = int(input(' Dame un número entero '))
        if n % 2 != 0:
            break
    suma = suma +n
    cont = cont +1
    if cont ==10:
        break
media = suma / 10
print (' la media de los 10 primeros números impares es :', media)
```

TRABAJO PERSONAL

Trabajo personal

Lectura recomendada

Cap 4, Apartado 4.2: Libro Introducción Programación Python (Andrés Marzal).

Laboratorio: Hoja Tema 1 parte 3.

Resolver ejemplos con Python empleando la sentencia repetitiva más adecuada:

1. Realizar producto de 2 enteros empleando únicamente la función suma. Modificarlo para incluir datos negativos

2. Dado un número, calcular su función de Fibonacci.

3. Dados 2 números introducidos por teclado se pide desarrollar un algoritmo que obtenga el mayor y el menor hasta que se introduzca 1111.

4. *Mostrar un cronometro con horas, minutos y segundos por pantalla..*