CASIO

Python con la calculadora gráfica CASIO fx-CG50

Autor | **Dr. Joan Verdaguer i Codina**

python™

Casio fx-CG50 con Python

Editado por: Casio Europa

CDU:
Deposito Legal: B 2020-
ISBN:

Índice general

Prefacio

Este libro ha sido escrito con el propósito de enseñar a utilizar la calculadora Casio fx-CG50 con programas escritos en el lenguage Python.

Los programas no solo se han probado en el emulador que suministra Casio para escribir y emularlos en el ordenador, también se han probado físicamente en la calculadora para verificar su correcto funcionamiento.

Ha de tenerse en consideración que se ha utilizado el software que viene de origen en dicha calculadora tal y como se suministra al futuro usuario una vez se ha abierto el envoltorio. Puede considerarse la versión v0.9 del libro Casio fx-CG50 con Python.

Los programas pueden escribirse directamente en el emulador para ver como funcionan con la calculadora virtual. También pueden escribirse en la libreta y guardarlo con la extensión .py y desde la calculadora virtual importar este fichero para ver su ejecución virtual.

Esta calculadora tiene la ventaja de poder ser utilizada en los exámenes sin que el alumno pueda acceder a cualquiera de los programas que hay insertados en ella. Esto se hace mediante el modo EXAMEN. Se sabe que la calculadora esta configurada en modo EXAMEN porque en el recuadro de la pantalla aparece un recuadro de color verde.


CASIO
fx-CG50 AU
COLOR GRAPHING
4.5 MB
Flash Memory
Preloaded Apps
USB cable included
python
Update to the latest OS.
https://edu.casio.com
Inspired by Australian Teachers for Australian Students

CAPÍTULO

1

Casio fx-CG50

Esta calculadora gráfica contiene un buen número de aplicaciones para ser usadas por el usuario.

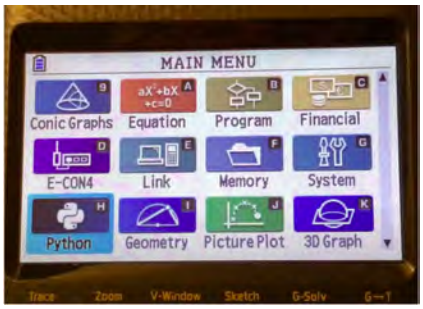
Estas aplicaciones se presentan en forma de iconos a los cuales se accede seleccionándolas con la tecla que hace las funciones de mouse.

Una de estas aplicaciones es la posibilidad de escribir, guardar y ejecutar programas escritos con el lenguaje Python.

Para utilizar Python hay que seleccionar el icono que lleva su nombre y una vez se ha seleccionado hay que pulsar la tecla **EXE** que está en la parte inferior derecha de la calculadora.

1.1 Teclas fundamentales

Todas las teclas son importantes en esta calculadora. Ahora bien, hay algunas específicas que se utilizan para escribir, y ejecutar los programas en Python.



1.1.1 Teclas F

Es el conjunto de teclas que hay inmediatamente despues de la panatalla las cuales estan nuemradas de izquierda a derecha del 1 al 6.

Una vez se ha pulsado el icono de Python aparece en la parte inferior de la pantalla los siguientes rótulos:

RUN	OPEN	NEW	SHELL	DELETE	SEARCH

Cada uno de estos rótulos tiene su correspondencia con los botones. De izquierda a derecha RUN corresponde a F1 y así sucesivamente hasta SEARCH que corresponde a F6.

1.1.2 Crear un programa

Para crear un programa hay que pulsar la F3 que tiene sobre ella la etiqueta NEW.

En la pantalla aparecen unos corchetes donde se ha de escribir el nombre del programa.

Atención! la tecla F5 es la que permite escribir en minúsculas, dado que por defecto el sistema está en mayúsculas. Se recomienda utilizar minúsculas.

Una vez escrito el nombre del programa hay que pulsar la tecla EXE. y automáticamente se entra en el editor del programa.

Aparecen al final de la pantalla las siguientes etiquetas

FILE	RUN	SYMBOL	CHAR	A<=>a	▷

1.1.3 Teclas de escribir

Para escribir hay que pulsar las teclas **SHIFT** y **ALPHA**. A partir de este momento se pueden utilizar las teclas que llevan el símbolo de cada letra para escribir el texto que corresponda.

1.1.4 Salir de editar un programa

Una vez escrito el programa para salir se pulsa las teclas **SHIFT** y **EXIT** apareciendo en la pantalla un recuadro con el mensaje siguiente:

Save this file?: Yes: [F1] No : [F6] Cancel: [AC]

El programa se guarda pulsando la tecla F1.

1.1.5 Ejecutar un programa

El guardar un programa aparece en la pantalla su nombre con la extensión **.py** que la genera la propia calculadora.

Al pulsar **F1** que implica ejecutar el programa (encima esta la etiqueta **RUN**) se entra en el **SHELL** del programa que es el equivalente a un terminal y se efectúa la entrada de datos que requiere el programa previamente guardado.

El programa no da errores si esta bien escrito y se cumplen las reglas de sintaxis de Python,

Una vez finalizado se pulsa la tecla **EXIT** para salir del **SHELL**.

CAPÍTULO

2

Python

Python es un lenguaje de programación de propósito general que es interpretado y que ha penetrado en diferentes ámbitos del quehacer humano dentro del mundo de la programación.

El gráfico anterior pretende dar unas razones por las cuales es bueno aprender Python. La número 6 se refiere a la placa Raspberry Pi que desde su origen lleva incorporado un interprete de Python para el aprendizaje de este lenguaje.

La Raspberry Pi originalmente fue diseñada para dotar a los alumnos de finales de secundaria y bachillerato de una herramienta sencilla, de bajo coste que pudieran adquirir y les motivara para estudiar la carrera de ingeniería informática; el como de su evolución y de su éxito no es objeto de este libro.

Hay muchos lenguajes de programación y todos tienen sus ventajas y sus inconvenientes, ahora bien la distinción principal que hay entre todos ellos es que son compilados o son interpretados.

A nivel de usuario, la distinción principal es que al ejecutar el programa no se pasa a la siguiente instrucción si la anterior es errónea.

Esta manera de ejecutar un programa permite visualizar en caso de error qué parte del programa escrito deja de funcionar, hacer la pertinente corrección y hacerlo rodar de nuevo. Esta facilidad en la depuración de los errores lo hace ideal para el aprendizaje de programación, en su uso desde finales de primaria hasta los cursos superiores.

Además, Python tiene un conjunto de librerías específicas de apoyo que sirven para ayudar a hacer programas de acuerdo con los requerimientos y las necesidades de la tecnología actual.

Los principales programas que apoyan un programa escrito en Python se muestran en el siguiente gráfico.

matplotlib	for visualization
SciPy	is a set of scientific and numerical tools for Python. It is built on top of NumPy.
NumPy	for scientific computing. Powerful N-dimensional array
pandas	is used for structured data operation and manipulations, data munging, and preparation data before to machine learning
learn	for performing all the machine learning activities

La versión que se utiliza es la de Python 3 porque el anterior (Python 2.7) ha dejado de ser soportada.

Se puede hacer una similitud del lenguaje Python como herramienta educativa con el lenguaje de programación BASIC utilizado en la década de los 60 hasta los 80 del siglo pasado. Este lenguaje también era interpretado y era la herramienta básica de los ordenadores de los años 70 y 80, por ejemplo el BBC micro empleado en las escuelas británicas.

El lenguaje BASIC era una simplificación del lenguaje FORTRAN que es compilado. Actualmente, FORTRAN es muy utilizado en dinámica de fluidos en supercomputación y fue el primer lenguaje utilizado por la NASA para hacer cálculos de órbitas de los satélites artificiales y misiones tripuladas como se muestra en la película Figuras Ocultas.

2.1 Comentarios

Los comentarios en un programa son importantes y hay que ponerlos para saber que hace el programa. Los comentarios dentro de Python van precedidos con el símbolo almohadilla #.

El ejemplo siguiente se refiere al programa suma.py y los comentarios están diferenciados por el color verde.

```
#coding:utf-8

#entrada de los datos
x =int(input("valor de la variable del número x= "))
y =float(input("valor de la variable del número y= "))
#cálculo de la operación
m=x+y
#visualización del resultado
```

```
9  print ("valor de la suma de x+y = ", m)
10 print()
```

NOTA: las líneas de cualquier programa escrito en Python no están numeradas.

Aquí están escritos para una mayor claridad de comprensión de cara al lector.

2.2 Reglas sintácticas

Además de las **reglas sintácticas y semánticas** propias de cada lenguaje, Python tiene unas reglas de estilo que hay que respetar cuando se escribe el código de un programa. Estas reglas tienen el acrónimo **PEP-8** y son conocidas como (`Style Guide for Python Code`).

En el ejemplo anterior se puede ver que la línea 2 esta en blanco. Las normas de estilo indican que ha de haber este espacio entre las librerías que se llaman y las líneas del programa.

Los programas escritos en Python tienen la extensión **.py**. A partir del capítulo 4 se verán diferentes ejemplos de programas con diferentes nombres però que tienen en común su extensión, p.e. `suma.py`, `calculadora1.py`.

2.2.1 Variables

Para introducir los datos en cualquier programa escrito en Python se utiliza la instrucción **input**.

Las variables deben estar precedidos de una de las instrucciones siguientes:

int cuando se introducen números enteros la instrucción **input** va precedida del acrónimo **int** que es la abreviatura de la palabra **integer**.

float si se ha de trabajar con números que tienen decimales la instrucción **input** ha de ir precedida de la palabra **float**; se escribe así porque los números decimales se representan en un formato denominado punto flotante.

str al trabajar con letras o caracteres alfabéticos se tiene una **cadena**, se llama así porque contiene una cadena de letras. El interprete de Python puede identificar las cadenas porque están encerradas entre comillas.

CAPÍTULO

3

Algorítmica básica

Pere Brunet[1], catedrático de Lenguajes y Sistemas Informáticos en la Universidad Politécnica de Cataluña, editó el libro titulado **Algorísmica bàsica** para la materia **Métodos Informáticos** que impartía en la escuela de ingenieros (ETSEIB) en marzo de 1988.

En el libro se define el concepto de algoritmo, y el tipo de estructuras algorítmicas básicas para una programación estructurada. Del contenido del libro se han adaptado los apartados que hay a continuación y en los capítulos siguientes.

3.1 Algoritmo y conceptos asociados

Algoritmo es una descripción no ambigua de los pasos (acciones) que hay que hacer para resolver cierto problema y llegar a una solución correcta en un tiempo finito.

En otras palabras, un algoritmo es un medio para comunicar la manera de resolver un cierto problema.

[1] http://www.lsi.upc.edu/ pere/

Se escribe un algoritmo porque hay una **necesidad**, primera condición del `Proyecto Tecnológico`; por tanto, la finalidad del algoritmo es que éste sea capaz de llegar a la solución del problema sin ninguna otra información y que ésta sea correcta.

Es evidente que el algoritmo no hace más que expresar la acción asociada a la resolución del problema planteado, en función de acciones elementales, que deben ser entendidas y ejecutadas por el interlocutor (computador). Por ejemplo, se quiere calcular la media aritmética de tres números a, b y c y nuestro interlocutor (computador) ya sabe cómo calcular medias, el algoritmo se limita a una única acción elemental:

Algoritmo cálculo de la media

algortmo *media*
Calcula la media aritmética de a, b, c
finalgoritmo

Como el computador sabe hacer las cuatro operaciones aritméticas es evidente que algunas de estas serán las acciones elementales. El algoritmo para resolver el problema describe la acción indicada *Calcula la media aritmética de a, b, c* en función de las acciones de sumar y dividir.

3.2 Análisis

Se llama **análisis** al proceso de razonamiento a seguir para llegar a algoritmos a partir del enunciado de un problema.

El analista es la persona que realiza el análisis y lo es porque esa persona tiene el conocimiento y la experiencia en saber representar con un algoritmo el problema planteado. Para ello el analista de sistemas utiliza técnicas de análisis y diseño para resolver problemas comerciales utilizando las tecnologías de la información.

3.2.1 Analista

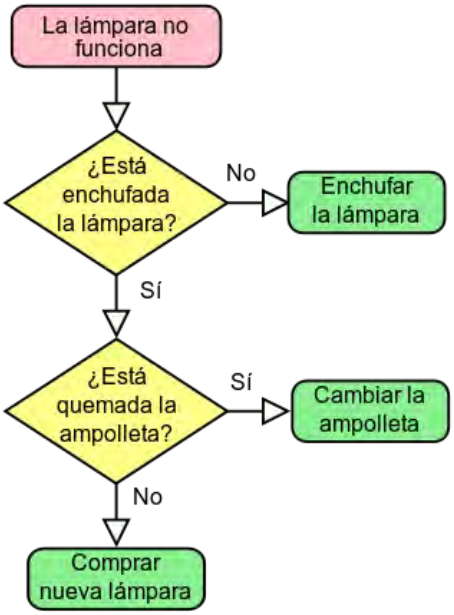
Los analistas de sistemas han de servir de agentes que identifican las mejoras organizativas necesarias, diseñan sistemas para implementar esos cambios y capacitan y motivan a otros para que usen los sistemas.

Si bien esta familiarizado con una variedad de lenguajes de programación, sistemas operativos y plataformas de hardware, normalmente no se involucra en el desarrollo real de hardware i/o software. También suele ser responsable de desarrollar análisis de costos, consideraciones de diseño, mejora del impacto del personal y plazos de implementación.

3.3 Representación de algoritmos

Un algoritmo se puede representar de diferentes maneras:

1. con un lenguaje textual, que puede ser de varios tipos:
   - Lenguaje de programación (Python, Java, C, C++, Javascript, etc.);
   - lenguaje natural (catalán, inglés, italiano);
   - pseudocódigo (describe con palabras la tarea que realiza el algoritmo)
2. con un lenguaje gráfico (diagrama de flujo **Flow Chart**), que consiste en utilizar símbolos gráficos que representan diferentes entidades lógicas en relación con la forma y permiten representar las acciones formando los diferentes símbolos de acuerdo con los diferentes modelos lógicos.



Este diagrama de flujo que se adjunta es un mal ejemplo pero es el que hay en la versión inglesa de la Wikipedia que se ha traducido al castellano tal y como se muestra.

El primer sistema para documentar el flujo de un proceso, fue presentado con el título *Gráficos de un proceso: primeros pasos para encontrar la mejor manera de hacer un trabajo*, en la Sociedad Americana de Ingenieros Mecánicos (ASME) en el año 1921. Fue en el año 1947 cuando ASME definió como estándar un conjunto de símbolos con el nombre de `Gráficos de procesos de operación y flujo`.

Goldstine y von Neumann utilizaron este estándar para planificar programas de ordenador en su informe no publicado, *Planificación y codificación de problemas para un instrumento de computación electrónica*.

3.3.1 Solución informática

Hay que recordar que hemos definido un **algoritmo** como un una secuencia de instrucciones perfectamente comprensibles y ejecutables de forma que, si se ejecuta en un orden específico y determinado, permiten solucionar un problema en un número finito de pasos.

Cuando la solución del problema debe ser informática, entonces hay que traducir del algoritmo a un **lenguaje de programación**, en nuestro caso Python, para que el ordenador pueda interpretar y ejecutar hasta llegar a la solución del problema.

Cada lenguaje tiene una serie de **reglas sintácticas y semánticas estrictas** a **seguir** para escribir un programa informático, las cuales describen la estructura y el significado respectivamente. Estas reglas permiten especificar tanto la clase de datos con que trabajará el programa como las acciones que realizará.

No cumplir con las **reglas sintácticas y semánticas** hace que el programa informático dé errores cuando se lo ejecuta.

CAPÍTULO

4

Teorema de Böhm-Jacopini

El profesor Brunet en su libro también explica que los esquemas de composición de un lenguaje informático para solucionar un algoritmo se dividen en tres tipos de estructuras:

1. Estructura secuencial: las instrucciones se ejecutan de acuerdo con el orden en que están escritas;

2. Estructura alternativa o selección o condición: hay una condición para evaluarse y dos posibles grupos de instrucciones a ejecutar; dependiendo del valor de la condición, se elige un bloque u otro;

3. Estructura repetitiva o iterativa o de ciclo: permite repetir un bloque de instrucciones, para un cierto número de veces, definido de acuerdo con el objetivo que se quiere conseguir.

Estas estructuras se conocen como el teorema de Böhm-Jacopini, enunciado en 1966 por dos informáticos italianos de los que toma su nombre.

El teorema afirma que: cualquier algoritmo se puede implementar utilizando sólo tres estructuras, la secuencia, la selección y el ciclo, que se aplicarán recursivamente en la composición de las instrucciones elementales.

En otras palabras, cualquier algoritmo se puede realizar utilizando sólo las tres estructuras indicadas.

Notese que estas estructuras cumplen con el principio básico de todo programa informático, que tiene una entrada (input) y una salida (output).

Además, cualquier programa informático normalmente es: la concatenación de uno de ellos, p.e. en el caso que solo sea secuencial serán diferentes bloques unitarios que conformaran un único bloque secuencial; o la combinación de estas estructuras, p.e. en el caso que se ejecute una parte secuencial, otra parte condicional, y a su vez otra parte secuencial.

CAPÍTULO

5

Estructura secuencial

Todos los programas de ordenador tienen una entrada (input) y una salida (output) y dentro de él hay el algoritmo.

Esto es lógico, utilizamos la calculadora para resolver un problema, por lo tanto una vez escrito el programa hay que entrar los datos (input) y visualizar en la pantalla el resultado obtenido (output).

5.1 Operaciones básicas

Las operaciones matemáticas elementales són la suma, la resta, la multiplicación y la divisón.

Estas cuatro operaciones se pueden hacer con cuatro programas independientes el uno del otro donde el principal objectivo es familiarizarse con la sintaxis de escribir un programa.

Se da como ejemplo el programa suma.py y se deja como ejercicio para el lector realizar los otros tres programas básicos. La diferencia que hay entre ellos es el signo de la operación, el nombre de la variable resultante y el texto que acompaña que ha de estar acorde a la operación que se va a realizar.

Es recomendable para quien se inicia en la programación hacer el diagrama de flujo pertinente asociado al programa.

5.1.1 Programa suma

Seguidamente se explica el programa textbf suma.py. El programa permite sumar dos números cualesquiera y este ejemplo de explicación es válido para las otras tres operaciones básicas (resta, multiplicación y división) que como se ha dicho anteriormente solo hay que hacer unos pequeños cambios.

El programa suma.py és un ejemplo de estructura de programa secuencial. En él hay una entrada, los dos números a sumar, y una salida, que es el resultado de la suma de estos números.

El programa comienza llamando el formato de codificación de caracteres utf-8. De hecho la versión de Python 3.7 ya no requiere esta instrucción; sin embargo, se ha creído conveniente citarla porque hay programas que tienen esta instrucción y el lector ha de saber a que hace referencia. Esta instrucción se debía poner siempre con la finalidad de evitar problemas con los acentos cuando se hace rodar el programa. Las diferentes lenguas europeas se distinguen principalmente porque varias de

ellas llevan acentos o tienen en su alfabeto letras que no están en el alfabeto de la lengua inglesa, la cual es la base del código ASCII en que se ha basado toda la informática.

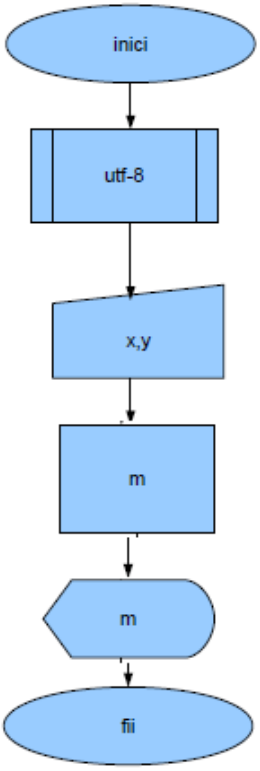
Siguiendo las normas de estilo PEP-8 la instrucción completa se pone en la cabecera del programa y se deja una linea en blanco.

La calculadora Casio fx-CG50 no requiere esta instrucción y por esta razón se ha omitido en el listado del código del programa, así como en los sucesivos.

El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo suma

Función (x,y)
m ← x+y
imprimir m

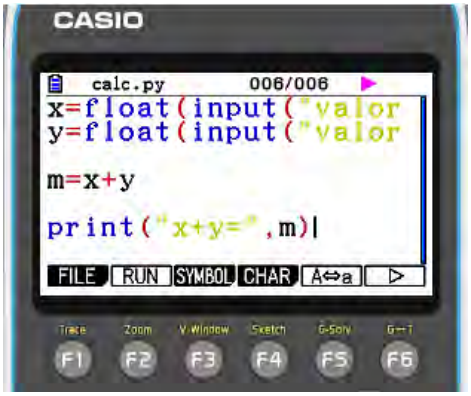


Se entran las dos variables utilizando la instrucción **input**. Estas variables si son números enteros deben estar precedidos de la instrucción **int**; si las variables son decimales deben estar precedidos de la instrucción **float**.

En este ejemplo se ha utilizado una de cada para visualizar su sintaxis. Seguidamente se hace la acción de sumarlos, el valor obtenido es visualiza mediante la instrucción **print** y el programa finaliza.

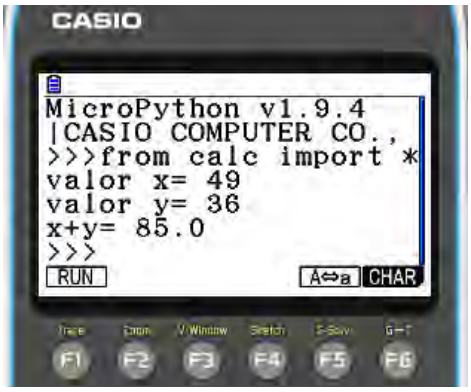
El código del programa **suma.py** es el siguiente:

```
#entrada de los datos
x =float(input("valor de x= "))
y =float(input("valor de y= "))
#cálculo
m=x+y
#visualización del resultado
print ("valor de x+y = ", m)
print()
```



Como se puede constatar este programa tiene la **estructura secuencial**, que es la más simple posible de las tres, y que cumple con el principio de ser un bloque donde hay una entrada y una salida.

Tal y como se ha explicado en un apartado anterior el diagrama de flujo es un lenguaje gráfico, que se utiliza para representar diferentes entidades lógicas mediante símbolos gráficos relacionados con su forma y esto permite representar las acciones.



El diagrama de flujo es una fiel representación, pero simplificada del código de programa escrito.

Los programas como LibreOffice (en este caso el draw) y el Word tienen estos símbolos y cuando se pone el símbolo del ratón sobre uno de los iconos sale su nombre asociado. Al ser LaTeX un lenguaje de programación que permite editar textos hay que escribir las pertinentes instrucciones que permitan visualizar el diagrama de flujo deseado.

En cualquier diagrama de flujos el símbolo de inicio y de final són iguales, este se puede hacer mediante un círculo o un óvalo y dentro se pone la palabra inicio o final según sea su cometido. Seguidamente se coge el símbolo de la primera instrucción del programa, en este caso es un **proceso predefinido** porque llamamos un programa ya que es el formato de codificación de caracteres utf-8; el símbolo es un rectángulo con dos barras verticales en su interior. Dentro de él se escribe la función que se llama.

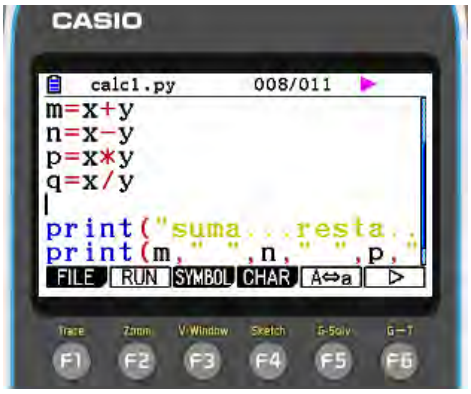
La siguiente instrucción en el programa es dar los valores de las variables y se utiliza el símbolo **entrada manual** y dentro se coloca las variables.

En el programa está el cálculo de la suma de sus variables. El símbolo que lo define se llama **proceso** y es un rectángulo. En él se escribe la variable que representa el resultado obtenido.

La última parte del programa es visualizar el resultado de la suma. El símbolo tiene la forma de una bala y este símbolo se llama **visualización o display**; dentro se escribe la variable que nos debe permitir ver el resultado obtenido.

5.1.2 Calculadora1

Efectuadas las cuatro operaciones separadamente ahora se hace el paso siguiente de realizar un programa al que se ha llamado **calc1.py** en que se debe entrar sólo una vez el valor de las dos variables y se ha de obtener el resultado de las cuatro operaciones.



El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo ca1culadora1

Función (x,y)
m ← x+y
n ← x-y
p ← x*y
q ← x/y
imprimir m, n, p, q

Este programa tiene una estructura secuencial; ahora bien en el hay cuatro bloques uno por cada operación.

El código del programa **calc1.py** es el siguiente:

```
from math import*

#entrada de los datos
x =float(input("valor x= "))
y =float(input("valor y= "))
#cálculo de las operaciones
m=x+y
n=x-y
p=x*y
q=x/y
#visualización de los resultados
```

```
print ("suma...resta...producto...division")
print (m," ", n," ",p, " ", q)
print()
```

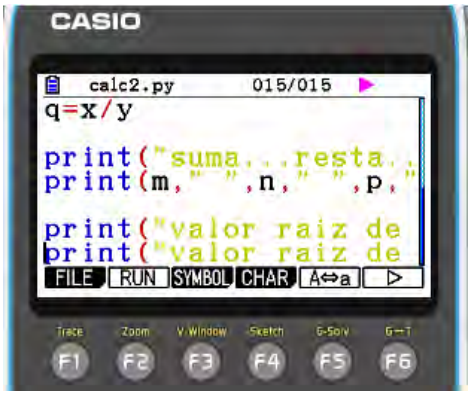
El diagrama de flujo es similar a los anteriores, la diferencia es que hay que escribir en el símbolo del **proceso** y también en el símbolo de **visualización** las cuatro operaciones sea escribiendo el nombre de las cuatro variables m, n, p y q.

5.1.3 Calculadora2

La diferencia con respecto al programa anterior es que además de las cuatro operaciones básicas se le ha añadido el cálculo de la raíz cuadrada.

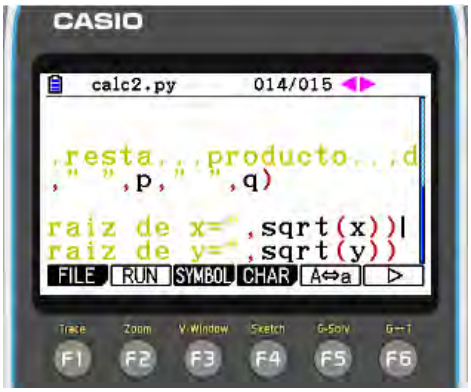
Este ejercicio es importante porque se introduce el concepto de librerías. Las librerías en Python son recursos con varias funcionalidades, de esta manera quienes escriben los programas evitan escribir el código por sí mismos al utilizar un código previamente escrito y bien definido. En este caso se evita escribir todo el código del programa que haría el cálculo de la raíz cuadrada.

Las normas de estilo obligan escribir el nombre de la librería en la parte superior antes de la línea en blanco que es donde empieza el código del programa.



La librería se llama **math** y la instrucción es **from math import***. Dentro de ella está el programa que calcula la raíz cuadrada de un número. La instrucción que hay que insertar en el código del programa para hacer dicho cálculo es **sqrt(x)**. La variable que se utiliza en el programa es la letra **x** i la letra **y**, por lo tanto se incluirá dicha instrucción para hacer idéntico cálculo con la segunda variable.

El algoritmo en pseudocódigo de este programa es el siguiente



Algoritmo calculadora2

Importar libreria math
Función (x,y)
m ← x+y
n ← x-y
p ← x*y
q ← x/y
imprimir m, n, p, q
sqrt(x)
sqrt(y)
imprimir raiz p, raiz q

El programa quedará escrito de la siguiente manera

```
from math import*

#entrada de los datos
x =float(input("valor x= "))
y =float(input("valor y= "))
#cálculo de las operaciones
```

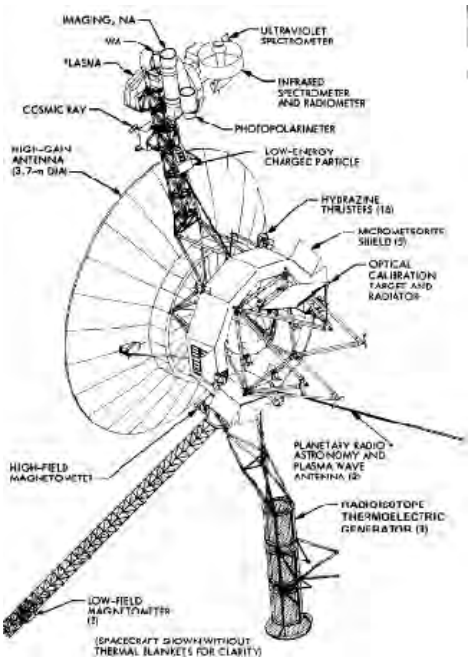
```
7  m=x+y
8  n=x-y
9  p=x*y
10 q=x/y
11 #visualización de los resultados
12 print ("suma...resta...producto...division")
13 print (m," ", n," ",p, " ", q)
14 #cálculo de las raíces cuadradas
15 print ("valor raíz x = ", sqrt(x))
16 print ("valor raíz y = ", sqrt(y))
17 print()
```

Lógicamente no todos los números que se han de introducir en cada una de las variables han de ser positivos y por esta razón se explica el apartado siguiente.

5.2 Unidades astronómicas

La unidad astronómica se utiliza cuando hay que hablar de la distancia de los planetas con la Tierra de la misma manera que se utiliza la velocidad de la luz para expresar dicha distancia.

Este es un programa de aplicación de la materia de astronomía que se enseña en primero de la ESO.



El enunciado es el siguiente: *La nave Voyager-1 fue lanzada por la NASA en el año 1977 y la última señal enviada para corregir la antena antes de salir definitivamente del sistema solar tardó en llegar 19 horas y 35 minutos. Se ha de calcular la distancia en km y en UA de la nave respecto de la Tierra.*

Para entender las dimensiones del satélite, el disco hace 3.7 metros de diámetro. Para enseñarlo a unos alumnos en la clase, es como unir 12 pies uno tras otro.

La resolución de este ejercicio obliga a buscar en Google el valor de la UA (unidad astronómica) y el valor de la velocidad de la luz, velocidad a la que viajan las señales electromagnéticas enviadas por la sonda espacial. Hay que insertar estos valores en el programa (líneas 7 y 9).

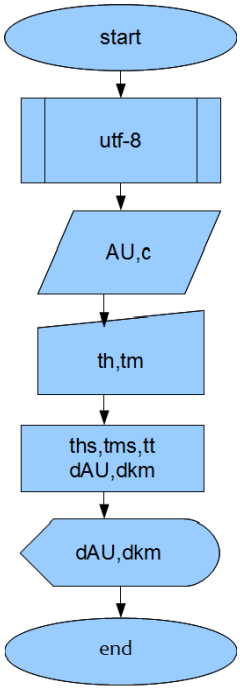
A continuación se introducen los datos. La instrucción **input** va precedida del **int** porque son números enteros. El programa los pasa a segundos (líneas 19 y 20) calculando el tiempo total (línea 22).

Sabemos que la velocidad (variable v) es la razón entre el espacio (variable x) y el tiempo (variable t) $v = \frac{x}{t}$. Se tiene el tiempo y la velocidad de la señal, que es la velocidad de la luz (c), el espacio recorrido vale $x = c * t$. El cálculo de la distancia se hace en la línea 24 y en las siguientes se hace la conversión a km y UA.

El algoritmo en pseudocódigo de este programa es el siguiente

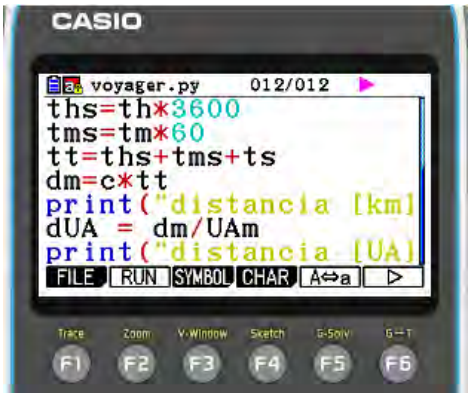
Algoritmo unidades astronómicas

Función (c,UA,th,tm,ts)
ths ← th*3600
tms ← tm*60
tt ← ths+tms+ts
dkm ← c*tt/1000
imprimir dkm
dUA ← dkm/UAm
imprimir dUA



Este programa se llama **vyg.py**, tiene una **estructura secuencial**. Su código es el siguiente:

```
print("Calcula la distancia de un satélite  artificial ")
print("respecto a la Tierra según tarda en llegar su señal")
print("dato c y UA")
c=float(input("vel. luz c[m/s]: "))
UAm=float(input("valor UA[m]: "))
print("Introducir el dato (separar horas, minutos y segundos")
print()
#introducir tiempo en horas
th=int(input("horas th= "))
#introducir tiempo en minutos
tm=int(input("minutos tm= "))
#introducir tiempo en segundos
ts=int(input("segundos ts= "))
#pasar los tiempos a segundos
ths=th*3600
tms=tm*60
#tiempo total en [s]
tt=ths+tms+ts
dm=c*tt
print("distancia [km]= ", dm/1000)
#cálculo de la distancia en UA
dUA=dm/UAm
print("distancia [UA]= ", dUA)
print()
```



En este diagrama de flujo aparece un icono nuevo que es un rectángulo inclinado. Este icono no se había utilizado hasta ahora porque no había ningún requerimiento de introducir datos fijos, como son en este ejemplo las constantes de la velocidad de la luz y del valor de la UA. Es recomendable consultar en Google los valores ambas constantes.

Seguidamente se entran los datos del tiempo que son una entrada manual; diferente a las utilizadas anteriormente pero válida para su representación gráfica. Finalizando el programa con la visualización de los datos solicitados.

La NASA dio como dato que el tiempo de recepción de la señal desde que la emitió el satélite Voyager I hasta su recepción en la estación terrestre fue de 19.35 horas. Esta recepción de la señal fue clave porque era la última vez que se podía interaccionar con ella para corregir la orientación de la antena y así seguir recibiendo señales hasta que se perdiera en el espacio.

EJERCICIO 1: Programas básicos

El objetivo de los ejercicios siguientes es familiarizarse en diseñar y escribir programas en Python.

E-1 Escribir un programa que haga la resta, el producto y la división de dos números cualesquiera. Se sugiere llamarlos sub.py, prod.py y div.py respectivamente.

E-2 Cuatro persones aficionadas a las adivinanzas forman un equipo siendo sus edades de 23, 39, 43 y 51 años. Realizar un programa que calcule la media de edad de este grupo.

E-3 Este programa es una simplificación del programa anterior.
Se ha dejado expresamente en su formato original para su resolución a nivel de código y de diagrama de flujo.

CodeWars 2019
Barcelona

7 A mirror on the Moon

3 points

Introduction

The Apollo 11 mission allowed mankind to step on the Moon 50 years ago. The astronauts Neil Armstrong and Buzz Aldrin, before the end of their moonwalk, installed a 2-foot wide panel studded with 100 mirrors pointing at Earth. It is called the "lunar laser ranging retroreflector array".

This array of mirrors allows to 'ping' the moon with laser pulses and measure the Earth-Moon distance very precisely. The laser pulse shoots out of a telescope on Earth, crosses the Earth-Moon divide, and hits the array. Then the mirrors send the pulse straight back where it came from. This allows, for example, to study the Moon's orbit. So, here is the simple formula to calculate this distance:

$$distance = \frac{speed\ of\ light * time\ taken\ for\ light\ to\ reflect}{2}$$

We would like to track the variability of the Earth-Moon distance so we need you to program this formula to find out the distance in kilometers. As you may know the speed of light is a constant value that is assumed as 300.000 kilometers / second. So, the parameter of this formula is the time taken for light to reflect. This value will be received as input expressed in milliseconds.

HINT: You do not need to mess with decimals to solve this formula.

Input

The time for the laser pulse to go and return expressed in milliseconds.

Output

The distance Earth-Moon expressed in kilometers.

Example

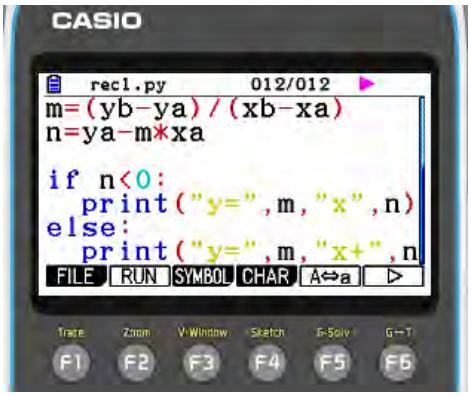
Input
2500

Output
375000

11

5.3 Ecuación de la recta

La ecuación general de la recta en forma explícita tiene la expresión matemática siguiente $y = m*x+n$ donde m es la pendiente de la recta y n es la ordenada del punto de intersección entre la recta y el eje de ordenadas. Si el parámetro n es diferente de cero implica que la recta no tiene origen en el punto (0,0) origen de los xy de coordenadas.

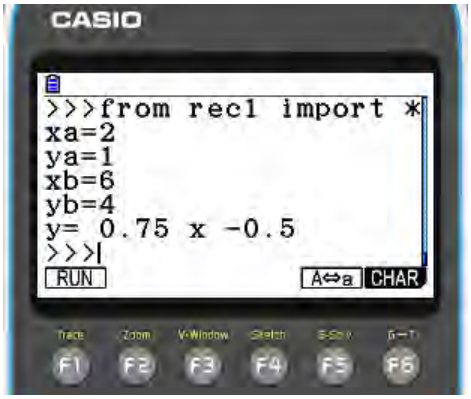


El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo ecuación de la recta

Función (xa,ya,xb,yb)
m← (yb-ya)/(xb-xa)
n ← xb-m*xa
Si n<0
..imprimir y=m*x-p
al contrario
..imprimir y=m*x+p

El programa se llama **rec1.py**, tiene una estructura secuencial con una estructura condicional al final y tiene el siguiente código de instrucciones




```
1  xa=float(input("xa: "))
2  ya=float(input("ya: "))
3  xb=float(input("xb: "))
4  yb=float(input("yb: "))
5  m=(yb-ya)/(xb-xa)
6  n=ya-m*xa
7  if n<0:
8    print("y=",m,"x",n)
9  else:
10   print("y=",m,"x+",n)
```

Puede probarse con las coordenadas siguientes A(2,1) y B(6,4) entre otras opciones.

Esta estructura condicional, que corresponde al capítulo siguiente, sirve para distinguir el signo del parámetro n en la expresión de la ecuación de la recta.

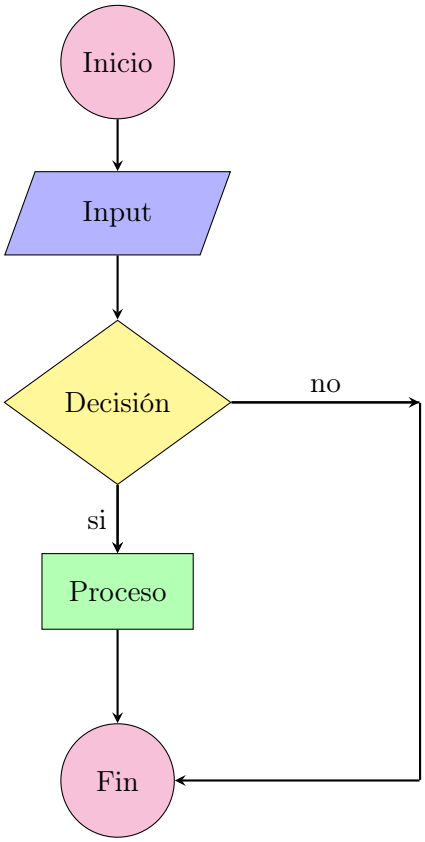
Hay que recordar que cuando se visualizan números si estos son positivos el programa no inserta su signo mientras que si son negativos les precede el signo menos. Esta es la razón por la cual en la línea 10 del programa se ve la variable `x` acompañada del signo `+` lo cual no ocurre en la línea 8 del programa al tener n un valor negativo.

CAPÍTULO

6

Estructura condicional

Una condición es planteada siempre en tomar una decisión sobre una propuesta, p.e. ir al cine o no ir, la decisión que se tome (*acción*) condicionará lo que se quiera hacer a continuación. De forma gráfica en términos generales y simples puede representarse con la gráfica de la derecha. Un ejemplo de esta condición es el siguiente programa



6.1 Sintaxis

La sintaxis básica para escribir la condición en Python es la mostrada a cntinuación.

if *condición*:
    (*acción*) aquí se escriben las líneas del programa

NOTA la sintaxis requiere que después de la condición se escriban los dos puntos **:** y que la siguiente línea tenga una sangría de 4 espacios y así todas las siguientes hasta que se termine la *condición*.

El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo condicional sin alternativa
Función (pregunta>10)
Si pregunta es no
..imprimir usar Casio fx-CG50

```
pregunta = str(input("eres <10a (s/n) = "))
if pregunta=="n":
    print()
    print ("Puedes usar la Casio fx-CG50")
print()
```

Puede existir una alternativa a la *condición* que obliga a escribir la instrucción **else**, de tal manera que el código se escribe de la siguiente manera

if *condición*:
    (*acción*) aquí se escriben las líneas del programa
else:
    (*acción*) aquí se escriben las líneas del programa

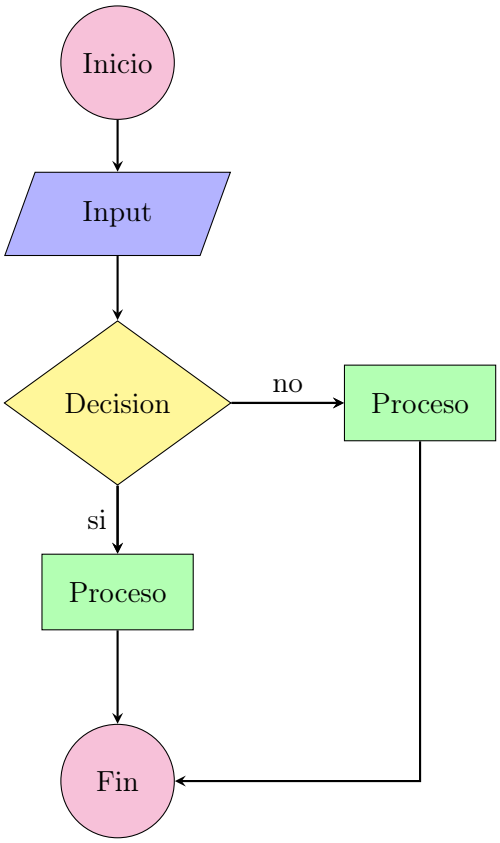
La instrucción **else** ejecutará la *acción* cuando no se cumpla nada de lo anterior.

Obsérvese que la instrucción **else** no lleva *condición* y que pasa a ejecutar directamente la *acción*, y hay que recordar que esta ha de estar alineada con la instrucción **if**.

El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo condicional con alternativa

Función (pregunta>10)
Si pregunta es no
..imprimir usar Casio fx-CG50
en caso contrario
..imprimir cálculo mental



El codigo del programa és el que se muestra a continuación.

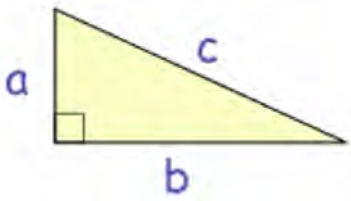
```
#coding:utf-8

pregunta = str(input("eres <10a (s/n) = "))
if pregunta=="no":
    print()
    print ("Puedes usar la Casio fx-CG50")
else:
    print()
    print ("Haz calculo mental")
print()
```

6.2 Pitágoras

La fórmula del teorema de Pitágoras define que en un triángulo rectángulo es la suma del cuadrado de cada cateto que es igual al cuadrado de la hipotenusa.

El cálculo de la hipotenusa o de uno de los catetos sirve para introducir la estructura condicional definida por el teorema de Böhm-Jacopini.

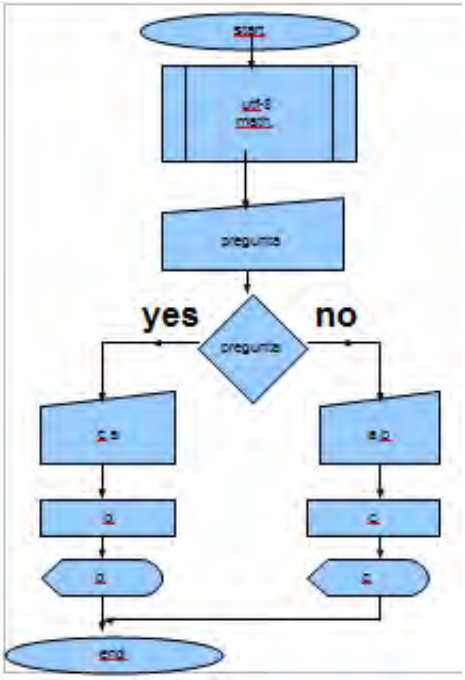
El teorema de Pitágoras relaciona los catetos (a, b) y la hipotenusa (c) mediante la expresión $a^2 + b^2 = c^2$



La condición sine quanon es que nos han de dar dos de las tres variables para calcular la tercera: si se quiere calcular la hipotenusa (c) se debe disponer del valor de los dos catetos (a,b); si se quiere calcular el valor de un cateto p.e. (b) se debe dar los valores de la hipotenusa (c) y el del otro cateto (a).

Como no puede existir un cateto o una hipotenusa de valor negativo el programa sólo llama la librería **math** necesaria para calcular la raíz cuadrada.

Aquí es donde interviene el analista porque la primera condición para escribir el algoritmo es saber hacer la pregunta o cómo hacer la pregunta para plantear el algoritmo. p.e. si se pregunta si se tiene un cateto implica añadir otra pregunta sque es si se tiene el otro cateto o no; en cambio si la pregunta es si dispone dela hipotenusa, entonces sólo hay que hacer una pregunta.



El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo Pitagoras
Importar librería math
Función (pregunta si/no)
Si pregunta es no
..función (a) ..función (b) ..c ← sqrt(a*a+b*b)
..imprimir c
al contrario
..función (a) ..función (c) ..b ← sqrt(c*c-a*a)
..imprimir b

El código del programa **ptg.py** es el siguiente:

```
1  #coding:utf-8
2  from math import sqrt
3
4  pregunta = str(input("hipotenusa (s/n) = "))
5  if pregunta=="no":
6      a=float(input("cateto a = "))
7      b=float(input("cateto b = "))
8      c=sqrt(a*a+b*b)
9      print ("hipotenusa c = ", c)
10 else:
11     a=float(input("cateto a = "))
12     c=float(input("hipotenusa c = "))
13     b=sqrt(c*c-a*a)
14     print ("cateto b = ", b)
15 print()
```

Al hacer la pregunta de si se tiene la hipotenusa, *línea cuatro* del código, la respuesta puede ser afirmativa o negativa. En la *línea cinco* se ha establecido la estructura condicional `if ....:` donde se da respuesta a la pregunta la que condiciona la estructura siguiente del código del programa. En caso afirmativo, las entradas de los datos deben ser el valor de la hipotenusa y uno de los catetos; en caso negativo, las entradas deben ser los valores de los dos catetos.

Como se ve en la *línea cinco* del código del programa se ha dicho de que no se dispone de la hipotenusa, en consecuencia en las líneas siguientes se pide el valor de los catetos y se ejecuta en la *línea ocho* se calcula el valor de la hipotenusa, que se visualiza en la siguiente línea del código.

La otra parte de la estructura condicional está en *línea diez* del código **else:** y el programa se ejecuta a partir de tener el dato de la hipotenusa y de un cateto para calcular el valor de el otro cateto, `línea trece` del código y seguidamente

visualizarlo.

El diagrama de flujo refleja gráficamente el programa que se inicia llamando el formato de codificación de caracteres utf-8 y la librería **math**. Seguidamente se hace la pregunta en que la instrucción **input** está precedida por el formato **str** porque la respuesta es una palabra y no un número. Seguidamente se establece la condición con la declaración **if-else**; primero, se escribe la declaración **if**, hay que recordar que las estructuras de control siempre terminan con el signo de puntuación de los dos puntos, seguidamente hay que escribir las instrucciones con una sangría de cuatro espacios hasta cuando se alcanza la declaración **else** que también acaba con el signo de puntuación de los dos puntos pero sín condición y, a continuación hay que escribir las instrucciones con una sangría de cuatro espacios hasta cuando se llega al final de la condición, en este caso es el final del programa.

EJERCICIO 2: Ejercicios de raíces números imaginarios y complejos

E-1 La relación entre los grados Celsius (centígrados) y los grados Fahrenheit viene dada por la expresión siguiente $\frac{C}{5} = \frac{F-32}{9}$
Realizar un programa que permita pasar de grados Celsius a grados Fahrenheit y viceversa.

E-2 Escriba este programa y ejecútelo entrando diferentes números para que visualice las diferentes opciones indicadas con el objetivo de analizar su funcionamiento.

```
#coding:utf-8

x =int(input("entrar un número menor que 25 x= "))\\
if(x <= 19):\\
    if(x == 19):\\
        print("x es igual que 19")\\
    else:
        print("x es menor que 19")
else:
    print("x es mayor que 19")
```

E-3 Cualquier tienda tiene una política de precios y descuentos.
Hay que definir como variables el precio, la cantidad, y el tipo de descuento. El IVA puede dejarse como constante, ahora bien en el caso hipotético de que se quiera comercializar el programa se recomienda dejarlo como una variable mas.
La empresa realiza descuentos sobre el importe pvp de esta manera: por importes superiores a 20 euros, se ha de incrementar el descuento en un 2 %; además para una cantidad no superior a 100 unidades se realiza un descuento adicional en un 1 %, si la cantidad no llega a 200 unidades se realiza un descuento del 2 %, para cantidades mayores el descuento adicional es del 3 %.
Hay que hacer un programa que cubra estas necesidades en que se detalle los descuentos aplicados y el importe total de la compra.

CAPÍTULO

7

Estructura iterativa

Las llamadas construcciones iterativas o repetitivas son construcciones de bucle . Python utiliza los nombres **for** y **while**. Ambas permiten realizar tareas repetitivas ayudando a iterar sobre cualquier objeto que precise ser iterado.

La cuestión obvia es que criterio hay que seguir para saber cuando utilizar la instrucción **for** y cuando utilizar la instrucción **while**.

Se recomienda utilizar la instrucción **for** cuando sepa el número fijo de veces que el ciclo debe repetirse.

Ahora bien, cuando no sepa cuántas veces debe repetirse el ciclo se recomienda utilizar la instrucción **while**.

Un analista sabe cuando es mejor utilizar una u otra instrucción porqué a veces se puede aplicar cualquiera de las dos instrucciones y hay que verificar cual de las dos es la mas óptima.

7.1 While

Hay muchos programas escritos con estructuras iterativas que utilizan la instrucción **while** aquí se muestran tres.

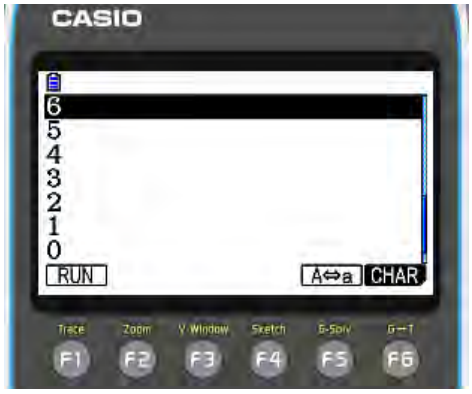
7.1.1 Substracción de números

Se quiere visualizar en orden descendiente los números empezando por el 10 hasta llegar al 0.

El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo decrecer un número

Definir a
Función mientras que a mayor que 0
..a ← a-1
..imprimir a



Las instrucciones para este programa son las que se muestran a continuación.

```
#coding:utf-8

a = 10
while a>0:
    a=a-1
    print(a)
```

El programa tiene la **condición** de calcular hasta que la variable **a** mientras sea mayor que cero (verdadero o true), y finalizará en el momento que el cálculo defina que la *condición* es falsa. Además cada vez dentro de la *acción* hace el cálculo y visualiza el resultado.

7.1.2 Suma de números

El programa que se muestra sirve para calcular la suma de números consecutivos empezando por el cero. Se ha limitado el número a los 10 primeros.

El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo suma números consecutivos

Definir a,b, maximo
Función mientras que a menor que maximo
..b ← b+a
..a ← a+1
..imprimir b

El código del programa **whL1.py** se muestra a continuación.

```
#coding:utf-8

a = 0
b = 0
maximo = 10
while (a <= maximo):
    b = b + a
    a = a+1
print ('suma =', b)
```

El programa da el resultado sin más y no muestra como ejecuta los pasos que son los siguientes:

- escribir las tres variables, se empieza por cero y el máximo ha de ser 10.
- dentro de la instrucción **while** se tiene la *condición*, y la *acción* es el bloque de código dentro del ciclo que se ejecuta siempre que la condición sea verdadera (true).
- dentro del bloque de *acción* sumamos a la variable **b** la variable **a** e incrementamos la variable **a** en 1.

- tan pronto como la variable **a** se convierte en 11, la *condición* dentro del bucle se evalúa como falsa (false) y, por lo tanto, el bucle deja de iterar.
- finalmente se imprime la suma de los primeros 10 números.

7.1.3 Fibonacci

La sucesión de Fibonacci es una sucesión matemática de números naturales donde cada uno de sus términos es igual a la suma de los dos anteriores. Cada uno de estos términos recibe el nombre de número de Fibonacci.

Tómese una sucesión de números naturales (cualquier número positivo a partir de cero) de tal forma que los dos primeros términos sean 0 y 1 y, que cada uno de los siguientes términos sea la suma de los dos anteriores.

Así esta sucesión quedará matemáticamente definida con las expresiones siguientes:

$$F(n)\begin{cases} 0, & \text{per n} = 0; \\ 1, & \text{per n} = 1; \\ F(n-1)+F(n-2) & \text{al contrario.} \end{cases}$$

El algoritmo en pseudocódigo de este programa es el siguiente

Algoritmo números Fibonacci

Funcion (n) Definir a, b, suma, paso
Función mientras que suma menor que n
..suma ← a+b
..a,b ← b, suma
..imprimir suma
..paso ← paso+1

El código del programa **fib.py** es el siguiente.

```
1  #coding:utf-8
2
3  n=int(input("hasta n= "))
4  a,b=0,1
5  suma=1
6  while(suma<n):
7      print (suma)
8      suma=a+b
9      a,b=b,suma
10 print()
```

En la línea 3 del programa se pide hasta que punto se ha de calcular la sucesión de los números de Fibonacci. Este número no es el número de iteraciones que el programa

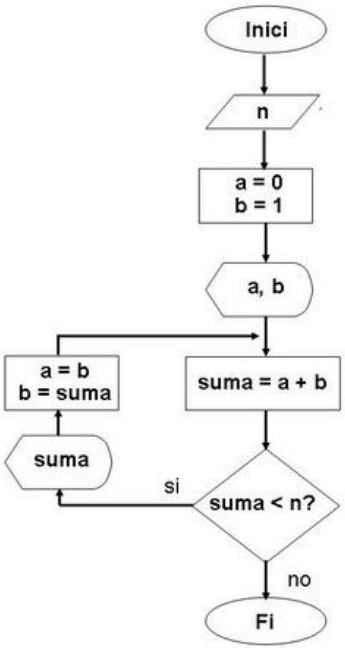
hará.

En la línea 4 se define el valor inicial de las variables **a** y **b**, En la línea 6 se inicializa el contador de ciclos que se necesitará para hacer la iteración.

En la línea 7 está la función **while** que inicia el bucle hasta mientras la condición sea cierta. La línea 10 tiene la instrucción que permite visualizar el valor cuando finaliza la iteración.

El diagrama de flujo refleja de forma gráfica como se realiza el cálculo de los números de Fibonacci.

El diagrama de flux del programa està en la pàgina següent. És l'exercici fet a classe manualment per tal d'ensenyar com funciona un bucle.



7.2 For

La instrucción **for** permite seguir iterando hasta el límite de la condición impuesta. Esta es la diferencia con respecto a la instrucción **while** en la que se sale de la iteración cuando la condición ha dejado de cumplirse.

7.2.1 Multiplicación

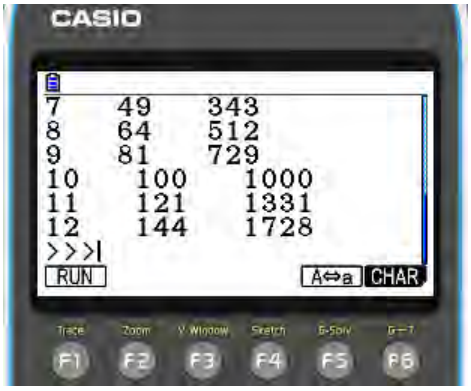
El algoritmo en pseudocódigo para la función `for` en este programa es el siguiente

Algoritmo multiplicación usando for

Función para todo i entre (limites menor y mayor)
Definir a, b, suma, paso
..imprimir i, i*i, i*i*i

Un ejemplo sencillo de aplicación de la instrucción **for** es el programa mostrado a continuación.

```
1  #coding:utf-8
2
3  for i in range(0,13):
4      print(i," ", i*i," ", i*i*i)
```



El programa coge el primer número calcula su cuadrado y su cubo visualizando el resultado, A continuación coge el siguiente número y procede de idéntica manera que con el número anterior y así sucesivamente hasta el límite que se ha señalado.

7.2.2 Suma de números2

En el apartado de la instrucción **while** se mostró un programa que calculaba la suma de números consecutivos empezando por el cero, limitándolo a los 10 primeros.

El algoritmo en pseudocódigo para este programa es el siguiente

Algoritmo suma usando for

Definir b
Función para todo a entre (limites menor y mayor)
..b ← b+a
..imprimir b

El código que se muestra a continuación hace lo mismo pero con la instrucción **for**. Al programa se le ha llamado **foL1.py** y se muestra a continuación.

```
#coding:utf-8

b = 0
for a in range (1, 11):
    b = b + a
print ('suma =', b)
```

El programa itera hasta el máximo rango que se le ha dado.

7.2.3 Cadena

Determinar cuando empieza un carácter concreto en una palabra puede realizarse mediante la instrucción **for**.

El ejemplo que se muestra a continuación detecta de la palabra *legitimar* la primera letra **i** parando el programa al existir la función **break**.

El algoritmo en pseudocódigo para este programa es el siguiente

Algoritmo busca una letra en una palabra

Funcion pal, car
Función para todo j en (pal)
..si j està en car
....salir
..imprimir j
imprimir fin

El programa se llama **busca.py** y su código se muestra a continuación.

```
print("**** busca letra ****")
pal=str(input("palabra: "))
car=str(input("letra: "))
for j in pal:
    if j == car:
        break
    print(j)
print("Fin")
```

Este programa puede modificarse para que muestre el resto de letras de la palabra. Simplemente hay que substituir la función **break** por la función **continue**.

Para hacer funcionar el programa se sugiere que detecte la letra **i** que existe en la palabra *legitimidad* y en función de una u otra instrucción muestra el resultado pertinente.

El algoritmo en pseudocódigo para este programa es el siguiente

Algoritmo busca la misma letra en todas las palabras

Funcion pal, car
Función para todo j en (pal)
..si j està en car
....continuar
..imprimir j
imprimir fin

En este caso el programa se llama **busca1.py** y su código es el siguiente.


```
print("**** busca letra ****")
pal=str(input("palabra: "))
car=str(input("letra: "))
for j in pal:
    if j == car:
        continue
    print(j)
print("Fin")
```

La instrucción **continue** puede substituirse por la instrucción **pass** en el caso de que se desee.

EJERCICIO 3: Ejercicios de for

E-1 En el programa **busca1.py** en lugar de introducir una palabra introduzca la frase *la legitimidad del insulto no existe* y para el carácter se sugiere buscar la letra **i**. Observe cual es el resultado.

E-2 Determine cual es la diferencia entre el programa **busca.py** y el programa **busca1.py**.

E-3 Realizar un programa que calcule el factorial de un número.

CAPÍTULO

8

Utilizando def

Una de las modalidades de programación en Python es utilizar la función **def** que es una palabra reservada.

La función **def** ha ganado muchos adeptos, no solo porque se puede llamar en cualquier parte de un programa o en otro programa, también porque en el mundo docente donde se puede evaluar el funcionamiento de un programa realizado por un alumno mediante el programa *Jutge.org*. El personal docente solo ha de escribir el nombre de la función para evaluarlo evitando así tener que copiar todo el código del programa. Esta facilidad de uso ha hecho muy popular el uso de la función **def**.

8.0.1 Sintaxis

La función def se puede escribir en cualquier lugar de un programa. Sin embargo, se recomienda escribirla en la primera línea.

Los pasos son

en la primera línea:

- palabra reservada def
  - nombre de la función. Es recomendable escribir todos los caracteres en minúsculas separando las palabras por guiones bajos [guía de estilo de Python]
  - paréntesis. Puede o no incluir los parámetros de la función, según se desee
  - dos puntos **:**

  en la línea siguiente:

- la **acción**, o sea las instrucciones que forman el código. A partir d'aquí la **acción** se escribe con sangría.
- se utiliza la declaración **return** para finalizar la ejecución de la **acción** y devuelve como resultado el valor de la expresión que sigue a la palabra **return** a la función que la llama. Las declaraciones posteriores a las declaraciones de devolución no se ejecutan. Si la sentencia return no tiene ninguna expresión, se devuelve el valor especial None. La declaración **return** se puede escribir al final, aunque no es obligatorio.

Nota: La declaración de devolución no se puede usar fuera de la función.

8.1 Ecuación de la recta

La ecuación general de la recta en forma explícita es $y = m * x + n$ donde m es la pendiente de la recta y n es la ordenada del punto de intersección entre la recta y el eje de ordenadas. Si es diferente de cero implica que la recta no tiene origen en el punto (0,0).

El algoritmo de este programa es el seguiente

Algoritmo ecuación de la recta

Función rec(xa,ya,xb,yb)
..Si xa=xb
....x=xa
..al contrario
....m← (ya-yb)/(xa-xb)
....n ← xb-m*xa
..Si n<0
....imprimir y=m*x-p
..al contrario
....imprimir y=m*x+p

El programa se llama **rec2.py** y tiene el siguiente código de instrucciones

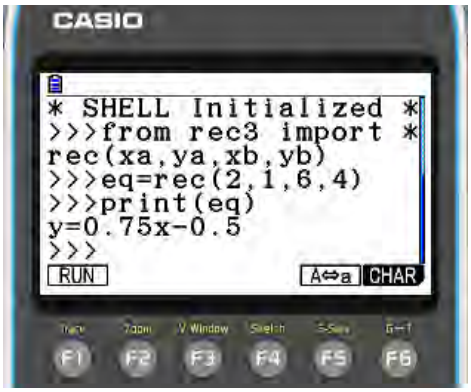
```
print("rec(xa,ya,xb,yb)")
def rec(xa,ya,xb,yb):
  if xa==xb:
    print("x=",xa)
  else:
    m=(ya-yb)/(xa-xb)
    n=ya-m*xa
  if n<0:
    print("y=",m,"x",n)
  else:
    print("y=",m,"x+",n)
```

Para ejecutarlo en el shell hay que escribir cuanto sale en la pantalla, por esta razón hay un `print` antes de la función `def` que es un recordatorio de cuanto hay que escribir **rec(xa,ya,xb,yb)**, siendo el punto A(xa,ya) y el punto B(xb,yb).

Notese que en el programa hay dos instrucciones `print` porque el valor de **n** puede ser positivo o negativo, si es positivo hay que poner el signo + y si es negativo es el programa quien pone el signo.

Otra opción es utilizar la instrucción **return**, y el programa se llama **rec3.py** y su código se muestra seguidamente.

```
print("rec(xa,ya,xb,yb)")
def rec(xa,ya,xb,yb):
  if xa==xb:
    print("x=",xa)
  else:
    m=(ya-yb)/(xa-xb)
    n=ya-m*xa
  if n<0:
    return "y="+str(m)+"x"+str(n)
  else:
    return "y="+str(m)+"x+"+str(n)
```



Para probarlo hay que ir al shell de la calculadora Casio y escribir lo siguiente:
eq = rec(2,1,6,4)
print(eq)

Dará idéntico resultado pero visualizado algo diferente; obsérvese que no hay espacios mientras que utilizando la instrucción **print** las comas hacen un espacio.

8.2 Números aleatorios

La función **random** permite generar números aleatorios entre otras de las funciones que tiene.

El programa en el ordenador elige un número aleatorio en un rango dado. Si se quiere obtener un entero aleatorio, se puede usar la función **randint**, la cual acepta dos parámetros: un número menor y uno mayor.

El algoritmo de este programa es el siguiente

Algoritmo números aleatorios

Importar librería random
Función dau(n)
..i ← [ ]
..para todo j a n
....i ← i+randint (menor, mayor)
..retornar i

El programa se llama dados.py y tiene el siguiente código de instrucciones

```
print("dau(n)")
from random import*
def dau(n):
  i =[]
  for j in range(n):
    i=i+[randint(1,6)]
  return i
```

Para ejecutarlo en el shell hay que escribir cuanto sale en la pantalla, por esta razón hay un `print` antes de la función `def` que es un recordatorio de cuanto hay que escribir **dau(n)**, siendo **n** la cantidad de números aleatorios que se desea obtener.

Hay que recordar que se ha limitado el número a 6 para simular las seis caras de un dado.

8.3 Estadística básica

En secundaria se enseña estadística básica en donde se aprende a calcular la media de una muestra y su desviación estándar.

El programa estad2.py El algoritmo de este programa es el siguiente

Algoritmo estadística básica

Función estad(*nmr)
..media ← 0
..suma2 ← 0
..media ← sum(m)/número items
..sumtot ← 0
..para todo i en m
....suma2 ← += $(i - media)^2$
..sigma ← raiz(suma2/numero items-1)
..retornar media, sigma

El programa se llama dados.py y tiene el siguiente código de instrucciones

```
print("stat([a,b,c,d,e])")
def stat(m):
    media=0
    suma2=0
    media = sum(m)/len(m)
    for i in m:
        suma2 += (i-media)**2
    sigma = (suma2/(len(m)-1))**0.5
    return(media,sigma)
```

Para ejecutarlo en el shell hay que escribir cuanto sale en la pantalla, por esta razón hay un `print` antes de la función `def`. Notese que en la notación de los datos a entrar hay que escribir obligatoriamente los paréntesis y los corchetes. P.e. se sugiere prara probar el programa escribir:

a=stat([1,2,3,4])
print("media,sigma =",a)

y mostrará el resultado de la media y la desviación estándar de los valores entrados.

CAPÍTULO

S

Ejercicios: Soluciones

S.1 Soluciones al capítulo 4

S.1.1 E-1

Resta

El código del programa **resta.py** es el siguiente:

```
#entrada de los datos
x =float(input("valor de x= "))
y =float(input("valor de y= "))
#cálculo
n=x-y
#visualización del resultado
print ("valor de x-y = ", n)
print()
```

Producto

El código del programa **prod.py** es el siguiente:

```
#entrada de los datos
x =float(input("valor de x= "))
y =float(input("valor de y= "))
#cálculo
p=x*y
#visualización del resultado
print ("valor de x*y = ", p)
print()
```

División

El código del programa **div.py** es el siguiente:

```
#entrada de los datos
x =float(input("valor de x= "))
y =float(input("valor de y= "))
#cálculo
q=x/y
#visualización del resultado
```

```
print ("valor de x/y = ", q)
print()
```

S.1.2 E-2

```
#entrada de los datos
a =int(input("edad a= "))
b =int(input("edad b= "))
c =int(input("edad c= "))
d =int(input("edad d= "))
#cálculo
m=(a+b+c+d)/4
#visualización del resultado
print("media edad = ", m)
print()
```

S.1.3 E-3

El código del programa **moon.py** es el siguiente:

```
#entrada de los datos
c =float(input("vel. luz [km/s]: "))
t =float(input("tiempo [s]:  "))
#cálculo
d=c*t/2
#visualización del resultado
print ("distancia [km]= ", d)
print()
```

S.2 Soluciones al capítulo 5

S.2.1 E-1

La relación entre los grados Celsius (centígrados) y los grados Fahrenheit viene dada por la expresión siguiente

$$\frac{C}{5} = \frac{F-32}{9}$$

La expresión para hallar los grados Celsius es la siguiente $C = \left(\frac{F-32}{9}\right) * 5$ y para calcular los grados Fahrenheit la expresión es $F = \left[\left(\frac{C}{5}\right) * 9\right] + 32$

Obviamente hay que hacer la pregunta clave: de cual temperatura se dispone.

A diferencia del programa de Pitagoras aquí no hay una opción clara y es a criterio del usuario optar por uno o por el otro.

El código del programa **grados.py** es el siguiente:

```
pregunta = str(input("tengo F (si/no) = "))
if pregunta=="no":
    C = float(input("valor grados C= "))
    F = ((C/5)*9)+32
    print ("grados F = ", F)
else:
    F = float(input("valor grados F= "))
    C = ((F-32)/9)*5
    print ("grados C = ", C)
print()
```

S.2.2 E-3

El código del programa **dcto.py** es el siguiente:

```
precio=int(input("precio= "))
cantidad=int(input("cantidad= "))
descuento=int(input("descuento= "))
iva=int(input("IVA= "))

# Calculos
if precio > 20:
    descuento = descuento + 2
if cantidad < 100:
    descuento = descuento + 1
elif cantidad < 200:
    descuento = descuento + 2
else:
    descuento = descuento + 3
euros_descuento = (( precio / 100 ) * descuento )
subtotal_precio = precio - euros_descuento
euros_iva = (( subtotal_precio / 100 ) * iva)
precio_final = subtotal_precio + euros_iva
total_factura = precio_final * cantidad

#*** Detalle factura ***
print("Precio: ",precio)
print("Cantidad: ",cantidad)
print("Dto: ",descuento)
print("Iva: ",iva)
print("Euros Dto: ",euros_descuento)
print("Precio con Dto: ",subtotal_precio)
print("Euros Iva: ",euros_iva)
print("Precio Final: ",precio_final)
print("Total: ",total_factura)
```

S.3 Soluciones al capitulo 6

S.3.1 E-1

Escribe toda las letras de la frase en columna eliminando todas las letras **i** que hay en la frase.

S.3.2 E-2

Este ejercicio se ha puesto para de mostrar la diferencia entre la instrucción **break** y la instrucción **continue**.

El programa **busca.py** interrumpe la búsqueda al encontrar la primera letra **i** de la palabra *legitimidad*. Esto ocurre debido a la instrucción break.

El programa **busca1.py** elimina todas las letras **i** de la frase pero la analiza en su totalidad. Aquí se demuestra el uso de la instrucción **continue**..

S.3.3 E-3

```
def factorial(num):
    n=num
    for i in range(1,num):
        n *= i
    return n

num = int(input("entrar número: "))
res=factorial(num)
print("El factorial de ",num," es ",res)
```

CAPÍTULO

N

Notas

CASIO®