

SUBINDICADOR # 2

Usa los literales en la definición de los tipos de datos, los operadores matemáticos y la función input() de Python, para su uso en el desarrollo de programas.

1. Introducción:

Los temas propuestos tienen por finalidad que los alumnos recuerden conceptos básicos asociados a los operadores matemáticos, reconozcan a las variables y constantes, los tipos de datos, entre otros; muy usados a lo largo del curso; así como también muestran de manera concreta como es la metodología de programación en todas sus etapas.

MÓDULO 2: Tipos de datos, variables, operaciones básicas de entrada y salida, operadores básicos

2. Los literales de Python.

a. Literales: Los datos en sí mismo

Los literales son aquellos datos cuyo valor lo determina, es decir le da un concepto, estos se relacionan con los tipos de datos simples que Python puede manejar y que son asignados a una variable como un valor que puede ser cadenas, enteros, flotantes, entre otros.

Si digitamos estas líneas:

```
print("Hola")  
print(2)  
print(True)
```

Al ejecutar se mostrará los valores indicados:

```
Hola  
2  
True
```

Ahora si agregamos lo siguiente:

```
print(type("Hola"))
print(type(2))
print(type(True))
```

Al ejecutar se mostrará los tipos de datos asociados (la clase a la que pertenece en Python):

```
<class 'str'>
<class 'int'>
<class 'bool'>
```

b. Los datos enteros, octales y hexadecimales

Las computadoras actuales trabajan con dos tipos de números, aquellos que tienen parte decimal y que se les denomina de punto flotante (en matemática los conocemos como números reales) y los que no tiene y que son denominados enteros.

Los números enteros se representan de manera directa:

3	Tres
67	Sesenta y siete
-8	Ocho negativo
1976	Mil novecientos setenta y seis
1_976	Mil novecientos setenta y seis
45998700	Cuarenta y cinco millones novecientos noventa y ocho mil setecientos
45_998_700	Cuarenta y cinco millones novecientos noventa y ocho mil setecientos

Podemos observar una particularidad respecto a los números enteros y es que cuando escribimos estamos acostumbrados a usar comas, por decir 45,998,700 esto no es posible en Python, pero si te permite usar un guion bajo si desea indicar una separación de miles, millones entre otros.

```
print(3947574)
print(3_947_574)
```

Al ejecutar se mostrará los valores indicados:

3947574**3947574**

Existen dos convenciones para el manejo de los números en Python para el manejo de números y son los octales (números comprendidos entre 0 y 7) y hexadecimales (números comprendidos entre 0 y 16).

0o65	Representación de un número octal
0x65	Representación de un número hexadecimal

Si llevamos ello a Python se observará lo siguiente:

```
print(0o65)
print(0x65)
```

Al ejecutar se mostrará los siguientes valores decimales respectivamente:

53
101

c. Los datos de tipo flotante

Son aquellos números que contienen decimales, a estos Python les denomina números de punto flotante, no se les puede representar con coma o punto decimal como en matemática, se les representa con punto decimal siempre.

Veamos algunos ejemplos

```
print(4.6)
print(3984.6)
print(-0.485)
print(.85)
print(9.)
```

Al ejecutar se mostrará los siguientes valores enteros respectivamente:

4.6
3984.6
-0.485
0.85
9.0

Puede ver algo interesante en los dos últimos ejemplos y es que los decimales puede expresarlos solo colocando un punto, Python sabe que se estará refiriendo al punto decimal.

d. Enteros vs flotantes

Se tiene algunas observaciones que deberá considerar al momento de trabajar y que resultan relevantes en la programación, cuando se escribe 4 estamos hablando de un número entero, pero cuando escribe 4.0 estamos hablando si bien es cierto del mismo valor numérico anterior, este es flotante(decimal).

Esto tiene que considerarlo, ya que en programación los termino de enteros y flotantes son distintos, por más que finalmente pertenecen a la misma familia de números.

Por otro lado, los flotantes también se pueden expresar en notación científica, cuando este resulta ser demasiado grande para su escritura, observar:

```
print(5)
print(5.0)
print(500000000000)
print(5E11)
```

Al ejecutar se mostrará los siguientes valores enteros respectivamente:

```
5
5.0
500000000000
500000000000.0
```

Puede observar en las impresiones como finalmente son tratado los números teniendo en cuenta los flotantes, algo que debe considerar es que los números que son tratados como notación científica, se devuelven como flotantes.

e. Codificando flotantes

Hemos observado que la notación científica nos permite describir número enormes, del mismo modo en caso de que los flotantes sean demasiado grandes se puede usar dicha notación, observar:

```
print(0.00000005)
print(45.364E-6)
```

Al ejecutar se mostrará los siguientes valores enteros respectivamente:

```
5e-08
4.5364e-05
```

De esa manera si deseamos expresar decimales con muchos ceros pues se deberá expresar como se indica.

f. Los datos de tipo cadena y su codificación

Las cadenas son tipos de datos que se deben expresar entre comillas, esto ocurre en todo lenguaje de programación (en algunos se usa los apóstrofes); de esa manera se le indica a Python que tenemos trabajando en una expresión, observemos:

```
print("Manejando el presupuesto 2030...!!!")
print("Muro de escudos")
print("El \"Heavy Metal\" es una variación del ROCK")
print("45")
print("0.67")
print("True")
```

Al ejecutar se mostrará lo siguiente:

```
Manejando el presupuesto 2030...!!!
Muro de escudos
El "Heavy Metal" es una variación del ROCK
45
0.67
True
```

Todo lo mostrado es de tipo cadena y es así como lo tratará Python, aunque visualmente algunos de los valores mostrados parecieran que no es así, sin embargo, en el código recordar que se ingresan entre comillas y eso es todo para que Python los reconozca como cadenas.

Por otro lado, observe el ejemplo en la tercera línea, puede ver que tiene incrustadas comillas internas, ello es gracias al uso de los backslash.

g. Los valores booleanos

Finalmente, los literales booleanos o lógicos son dos, True que significa verdadero y False que es falso.

El nombre booleano, proviene de George Boole (1815-1864), el autor de Las Leyes del Pensamiento, las cuales definen el Álgebra Booleana; que se denota verdadero como 1 y falso como 0.

Algo adicional es que, entre la falsedad y la verdad, al analizarlos de manera independiente siempre prevalece la verdad.

3. Operadores y Herramientas para la manipulación de datos

a. Las variables, nombrar variables de manera correcta, creación y uso

La variable es un espacio, que permite la asignación de diversos valores (literales), las variables deben tener un nombre adecuado y un valor con el que trabaje.

Se tiene las siguientes reglas para nombrar a una variable:

- El nombre de la variable debe de estar compuesto por MAYÚSCULAS, minúsculas, dígitos, y el carácter _ (guion bajo)
- El nombre de la variable debe comenzar con una letra
- El carácter guion bajo es considerado una letra
- Las mayúsculas y minúsculas se tratan de forma distinta, es decir Python diferencia entre ellos, por decir si nombramos una variable con el nombre de Carro y otra como CARRO, Python define que ambas variables son distintas
- Los nombres de las variables no pueden ser igual a alguna de las palabras reservadas de Python

Cuando se nombra una variable, no puede usarse estos nombres, ya que son palabras reservadas para Python:

False, None, True, 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

b. Asignación de valores a variables

Para que una variable exista, debe asignársele un valor, para ello solo es necesario colocar un símbolo igual (=) y otorgarle el valor deseado, observar:

```
X=10
print(X)
Nom="Eduardo"
Nota1=17
Nota2=19
print("El alumno ",Nom)
print("Tiene las siguientes notas: ",Nota1,Nota2)
```

Al ejecutar se mostrará lo siguiente:

```
10
El alumno  Eduardo
Tiene las siguientes notas:  17 19
```

Observar cómo se está nombrando a las variables y como se manipula una vez que se le asignó un valor.

Al momento de asignar valores a una variable, puede ocurrir algo que en ningún otro lenguaje de programación ocurre y es lo siguiente:

```
Nt=80
print(Nt)
Nt=3
print(Nt)
Nt="Los valientes mueren de pié...!!!"
print(Nt)
Nt=20.6
print(Nt)
```

Al ejecutar se mostrará lo siguiente, sin la existencia de ningún error:

```
80
3
Los valientes mueren de pié...!!!
20.6
```

Esas asignaciones observadas no podrían ser aceptadas en ningún otro lenguaje de programación, sin embargo, en Python si se puede trabajar de ese modo por una sencilla razón que una variable no requiere declaración (es decir asignarle un tipo de dato para crearla), esta es una ventaja en muchos aspectos, pero requiere de mucho control en el código cuando se programa, ya que puede terminar quedando alguna variable sin asignación y generar error.

c. Resolviendo problemas matemáticos: operaciones combinadas

Para resolver operaciones aritméticas se debe conocer cuáles son los operadores que reconoce Python, a continuación, se muestran en la tabla:

Operador	Descripción	Ejemplo
+	Suma	$r = 3 + 2$ # r es 5
-	Resta	$r = 4 - 7$ # r es -3
-	Negación	$r = -7$ # r es -7
*	Multiplicación	$r = 2 * 6$ # r es 12
**	Exponente	$r = 2 ** 6$ # r es 64
/	División	$r = 3.5 / 2$ # r es 1.75
//	División entera	$r = 3.5 // 2$ # r es 1.0
%	Módulo	$r = 7 \% 2$ # r es 1

Todas las operaciones simples o combinadas que incluyen a estos operadores se rigen por las reglas de precedencia establecidas por las matemáticas:



A continuación, veremos el desarrollo paso a paso de 2 ejercicios de operaciones combinadas:

$$\begin{aligned}
 &10 + 10 * 10 - 10 / 2 \\
 &10 + 100 - 10 / 2 \\
 &10 + 100 - 5.0 \\
 &110 - 5.0 = \mathbf{105.0}
 \end{aligned}$$

$$\begin{aligned}
 &2 * 4 + ((7 * 8) \% 3) \\
 &8 + (56 \% 3) \\
 &8 + 2 = \mathbf{10}
 \end{aligned}$$

Para validar el resultado en Python solo se tendrá que incluir la operación dentro de la función print() como se muestra:


```
print (10+10*10-10/2)
print (2*4+ ( (7*8) %3) )
```

Al ejecutar se mostrará directamente el resultado:

```
105.0
10
```

Algo que debe considerar es que si en la operación combinada se llega a encontrar una división real o alguna operación que tenga componentes flotantes(reales), el resultado final siempre será flotante.

OBS: Transformando expresiones algebraicas:

Cuando diseñamos o traducimos operaciones combinadas que provienen de ecuaciones existentes o planteadas por terceros, debemos considerar que estas expresiones algebraicas en general, estén traducidas en expresiones lineales, veamos:

$$(2B + 3) + 4R + \frac{6+P}{7}$$

$$(2 * B + 3) + 4 * R + (6 + P) / 7$$

$$A + B + C + \frac{4^R + 5^A}{E + A} + C$$

$$A + B + C + (4 ** R + 5 ** A) / (E + A) + C$$

De ese modo, Python podrá operar la operación combinada, siempre en cuando las variables tengan asignados valores previos.

d. Operadores abreviados

Estos operadores permiten al programador, hacer que algunas operaciones o instrucciones se hagan de manera más fácil, debe tener en cuenta que siempre se ubican antes del signo igual.

Operador	Ejemplo	Equivalencia
<code>+=</code>	<code>x += 2</code>	<code>x = x + 2</code>
<code>-=</code>	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	<code>x *= 2</code>	<code>x = x * 2</code>
<code>/=</code>	<code>x /= 2</code>	<code>x = x / 2</code>
<code>%=</code>	<code>x %= 2</code>	<code>x = x % 2</code>
<code>//=</code>	<code>x //= 2</code>	<code>x = x // 2</code>
<code>**=</code>	<code>x **= 2</code>	<code>x = x ** 2</code>
<code>&=</code>	<code>x &= 2</code>	<code>x = x & 2</code>
<code> =</code>	<code>x = 2</code>	<code>x = x 2</code>
<code>^=</code>	<code>x ^= 2</code>	<code>x = x ^ 2</code>
<code>>>=</code>	<code>x >>= 2</code>	<code>x = x >> 2</code>
<code><<=</code>	<code>x <<= 2</code>	<code>x = x << 2</code>

e. Generando comentarios entre sentencias de programación

Cuando se tiene un programa codificado, puede existir instrucciones o comentarios que se deseen dejar registrados para que sirvan de guía a los programadores que trabajan en el proyecto o simplemente sea una ayuda memoria para hacer indicaciones a partes del código que se consideran un poco complejas.

Por ello se hace uso de caracteres especiales que pueden inhabilitar una línea o un conjunto de líneas de programación, estos caracteres son el símbolo `#` y los símbolos `'''` juntos, observar:

```
print("Ingrese un nombre:")
Nom=input()
#La línea anterior permite el ingreso
#del nombre del usuario
print("Ingrese la edad:")
Ed=int(input())
#La línea anterior permite el ingreso
#de la edad del usuario
print(Nom, "y su edad es :",Ed)
'''
El programa anterior permite
ingresar el nombre y la edad
de un cliente
'''
```

Al ejecutar se mostrará directamente el resultado:

```
Ingrese un nombre:
Juan
Ingrese la edad:
25
Juan y su edad es : 25
```

Podrá observar que al colocar un solo símbolo de numeral (#) toda la instrucción no se ejecuta, puede también marcar un conjunto de sentencias y presionar las teclas Alt+3 y se convertirá en comentario, para retirar el comentario puede borrarlos de manera individual o en su defecto presionar Alt+4.

Del mismo modo, si desea inhabilitar un conjunto de instrucciones directamente, puede colocar tres apostrofes desde donde desea bloquear y volver a colocar tres apóstrofes donde es el final del bloqueo (ver ejemplo).

4. Como interactuar con el computador

a. La función input()

Esta función es usada para capturar los valores ingresados por los usuarios en la consola, para que un programa funcione requiere de datos (valores), ya que, sino no tiene sentido el programa no generaría información.

La función input(), puede tener parámetros que permiten colocar títulos, sin la necesidad de colocar a print() antes para hacer comprensivo lo solicitado a ingresar.

```
print("Ingrese un Producto:")  
Nom=input()  
Pr=input("Precio:")
```

Al ejecutar se mostrará de la siguiente manera, donde se solicita dos datos, uno por cada input():

```
Ingrese un Producto:  
Camote  
Precio:2
```

b. Argumentos y operaciones prohibidas

Cuando se usa la función input() de manera directa, este captura lo ingresado por el usuario en la consola, y automáticamente le asigna el tipo cadena (str), por lo que si desea realizar alguna operación matemática no podrá y saldrán errores.

c. Conversión de datos o casting

Hacer un *cast* o *casting* significa convertir un tipo de dato de un literal a otro, se tienen diferentes tipos de *cast* o conversión de tipos que se pueden hacer.

- **Conversión implícita:** es realizada automáticamente por Python. Sucede cuando realizamos ciertas operaciones con dos tipos distintos.
- **Conversión explícita:** es realizada expresamente por nosotros, para ello se hace uso de las funciones str(), int(), float().

La conversión explícita es la que usamos para poder manipular los valores de manera adecuada dentro de un programa, la conversión implícita ocurre al momento de asignar valores a variables.

En el ejemplo, vemos que antes de realizar la operación de sumar X1 con X2, se convierte a entero, sino saldría error, ya que como dijimos antes la función input() captura literales tipo cadena.

```
print("Ingrese 1er número:")  
X1=input()  
print("Ingrese 2do número:")  
X2=input()  
S=int(X1)+int(X2)  
print("La suma es:",S)
```

En el ejemplo, vemos que antes de realizar la operación de sumar X1 con X2, se convierte a flotante, sino saldría error, ya que como dijimos antes la función input() captura literales tipo cadena.

```
print("Ingrese 1er número:")
X1=input()
print("Ingrese 2do número:")
X2=input()
S=float(X1)+float(X2)
print("La suma es:",S)
```

Finalmente, en este último ejemplo, vemos que se está aplicando el casting en la lectura de las variables, algo que es adecuado manipular.

```
X1=int(input("Ingrese 1er número:"))
X2=int(input("Ingrese 2do número:"))
S=X1+X2
print("La suma es:",S)
```

d. Operaciones con cadenas: concatenar, replicación

Cuando se trabajan con cadenas, podemos usar algunos operadores matemáticos que se interpretan de manera distinta a lo que conocemos en las operaciones matemáticas, observar:

```
Nom="Eduardo"
Edad="40"
Sexo="Masculino"
print(Nom+Edad+Sexo)
print(Nom+" "+Edad+" "+Sexo)
```

Al ejecutar se mostrará lo siguiente:

```
Eduardo40Masculino
Eduardo 40 Masculino
```

Como podrá observar cuando usamos el símbolo + con cadenas, estas se concatenan es decir se unen.

Observe ahora este ejemplo:

```
print("Sistemas")
print("Sistemas"*3)
```

Al ejecutar se mostrará lo siguiente:

```
Sistemas
SistemasSistemasSistemas
```

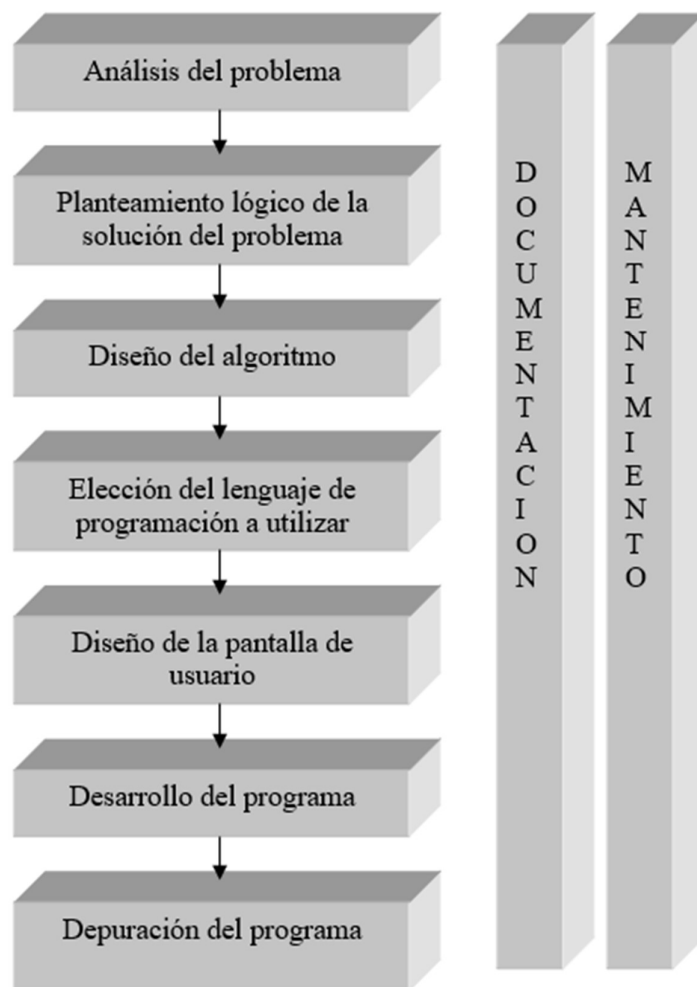
Como podrá observar cuando usamos el símbolo * con cadenas, estas se multiplican y se muestran tantas veces repetido como número se indicó.

5. Metodología de programación

Las personas entienden el mundo construyendo modelos mentales de porciones del mismo, de esta manera toman los elementos importantes dejando los detalles irrelevantes fuera del modelo.

En el proceso de diseño de software se siguen algunos pasos que permiten obtener la resolución del problema partiendo de un modelo simplificado del mismo.

Las etapas en la resolución de un problema **GENERAL** en el ámbito computacional se presentan en la siguiente gráfica, estas muestran en detalle cómo es la formalidad para el desarrollo de programas de cualquier género:




a. Análisis

En esta etapa se busca los elementos que alimentan el proceso (especificaciones de entrada), los elementos (datos) una vez procesados van a generar información (especificaciones de salida).


Para lograr encontrar esos datos iniciales, basados en el problema que se requiere resolver, se debe realizar dos preguntas, que permitirán iniciar el proceso de solucionar el problema planteado:

- **¿Qué me piden?** – Es el problema central, que puede tener varias aristas.
- **¿Qué necesito?** – Son los datos que requiere para resolver lo indicado anteriormente.

	<h4>Aplicación Práctica</h4> <p>Diseñar un programa que permita ingresar un número de 2 cifras de la forma ab, luego mostrar la suma de sus componentes (dígitos)</p> <p>Análisis del problema.</p> <ul style="list-style-type: none">• ¿Qué me piden? <p>Me están pidiendo hallar la suma de los dígitos que componen un número de 2 cifras.</p> <ul style="list-style-type: none">• ¿Qué necesito? <p>Requiero conocer el número de 2 cifras.</p>
--	---


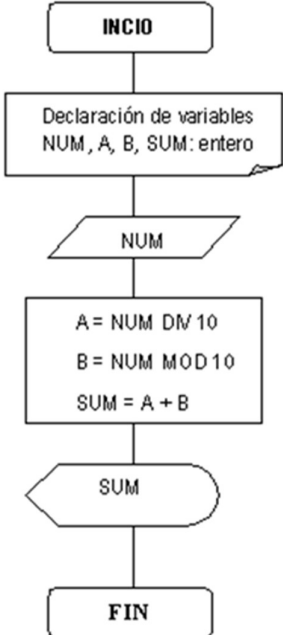
b. Planteamiento lógico

El planteamiento lógico de basa en la idea que uno tiene para resolver el problema, basado en un modelo matemático o secuencia de procesos (leer datos, registrar datos, consultar datos, reportar datos, etc.).

	<h4>Continuación de la aplicación práctica</h4> <p>Para poder obtener la suma de los dígitos del número de 2 cifras que se ingrese deberemos de desdoblarlo en sus componentes es decir obtener las unidades y decenas, esta operación de desdoblamiento se puede realizar de diversas formas, la mas común:</p> <ul style="list-style-type: none">• Dividir el número entre 10 (recordar que manejamos el sistema decimal).• De la división obtener el cociente entero y el residuo en variables distintas.• Luego aplicar la formula siguiente: <p>$\text{Suma} = \text{cociente entero} + \text{residuo}$</p> <p>Finalmente se obtiene la suma de los dígitos.</p>
---	--

c. Diseño: Herramientas para Diseñar Algoritmos: pseudocódigos y diagramas de flujo, simbología

En esta etapa se aplican técnicas de programación tales como el diseño descendente, refinamiento por pasos, obteniéndose la representación del algoritmo solución. Para ello se utilizan herramientas como el diagrama de flujo o el pseudocódigo.

	<p>Continuación de la aplicación práctica</p> <p>Usaremos las herramientas básicas para el desarrollo y diseño de Algoritmos:</p> <p>Pseudocódigo</p> <p>INICIO</p> <p>Declaración de variables</p> <p>NUM, A, B, SUM: entero</p> <p>Escribir ("Ingrese el valor del numero de 2 cifras:")</p> <p>Capturar(NUM)</p> <p>$A = \text{NUM DIV } 10$</p> <p>$B = \text{NUM MOD } 10$</p> <p>$SUM = A + B$</p> <p>Mostrar ("La suma de los dígitos es:",SUM)</p> <p>FIN</p>
	<p>Diagrama de Flujo</p> 

▪ **Pseudocódigo**

Es una lista detallada de las acciones que se deben realizar para escribir un programa. En esta herramienta de programación las instrucciones se escriben en palabras similares al español. Aunque no existen reglas para la escritura del pseudocódigo en español, se ha recogido una notación estándar que se utiliza en la mayoría de libros de metodología de la programación, observar:

Pseudocódigo

INICIO

Declaración de variables

A, B, X: entero

Escribir('Ingrese el valor de A:')

Capturar(A)

Escribir('Ingrese el valor de B:')

Capturar(B)

$X = 3 * A + B$










Mostrar('El valor de x en la ecuación es:', X)

FIN

▪ **Diagrama de Flujo (DFD)**

Es un diagrama secuencial empleado para mostrar los procedimientos detallados que se deben seguir al realizar una tarea. Los diagramas de flujo se usan normalmente para seguir la secuencia lógica de las acciones en el diseño de programas de computadoras. Su construcción es importante porque a partir de él se escribe un programa en algún lenguaje de programación.

La simbología que se muestra es la estándar, sin embargo, en el tiempo los autores de herramientas CASE, han modificado alguno de ellos, significando lo mismo en la interpretación general.

Representación del símbolo	Descripción
	Este símbolo marca el inicio y fin de un diagrama de flujo.
	Usaremos este símbolo para representar la declaración de variables.
	Se utiliza para definir los datos de entrada.
	Representa un proceso, dentro de él se definen asignaciones y expresiones de cálculo en general.
	Este gráfico mostrará la información resultante (el resultado final).
	Las flechas o líneas indican la dirección del flujo del diagrama.
	Se utiliza para expresar conexión dentro de una misma página.
	Símbolo utilizado para expresar un módulo de un problema
	Símbolo utilizado para expresar conexión entre páginas diferentes

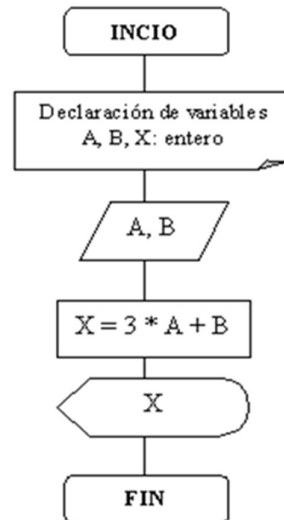
Reglas para la construcción de diagramas de flujo

Como los diagramas de flujo ilustran gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema. Los símbolos utilizados deben colocarse adecuadamente, para crear una estructura flexible que ilustre los pasos a seguir para alcanzar un resultado correcto.

- Todo diagrama debe tener un inicio y un fin
- Las líneas utilizadas para indicar la dirección del flujo del diagrama, deben ser rectas verticales o horizontales. No se puede incluir líneas diagonales, ni líneas cruzadas.
- Todas las líneas utilizadas como conectores deben partir de un símbolo
- El diagrama de flujo debe ser construido de arriba abajo (top-down) y de izquierda a derecha (right-left)
- La notación utilizada en el diagrama de flujo debe ser independiente del lenguaje de programación.
- Recomendamos agregar comentarios cuando utilice una expresión compleja para ayudar a entender su significado.
- Si el diagrama de flujo requiere más de una hoja para su construcción, utilice el conector de páginas diferentes y numere las páginas secuencialmente

Observemos el ejemplo siguiente:

Diagrama de Flujo



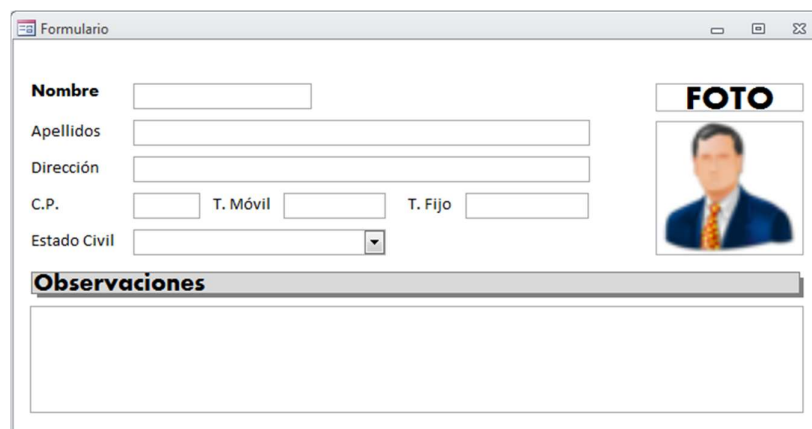
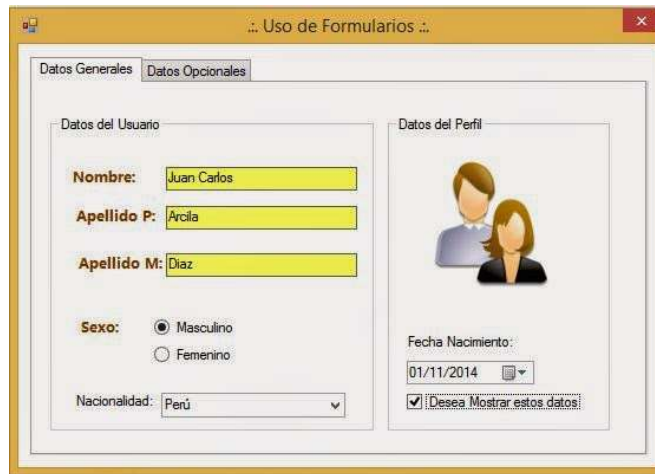
d. Lenguaje de programación

En esta etapa se deberá escoger el lenguaje sobre el cual trabajaremos, de modo que se pueda observar cómo se transformará el algoritmo planteado anteriormente en un lenguaje de programación.



e. Diseño de la interfaz

El formulario es el principal medio de comunicación entre el usuario y la aplicación ya que los usuarios interactúan con los controles que están sobre el formulario, a esta interfaz se le denomina GUI (Guía de Interfaz de Usuario).



f. Desarrollo y depuración del programa

Una vez diseñada la GUI, queda hacer que esta sea efectiva e inicie el proceso de captura o visualización de datos, dependiendo la naturaleza del programa a desarrollar.

```
print("Ingrese un número de 2 cifras:")
Num=int(input())
A=Num//10
B=Num%10
Sum=A+B
print("La suma es:", Sum)
```

Al ejecutar, el código devolverá lo siguiente:

```
Ingrese un número de 2 cifras:
34
La suma es: 7
```

6. Actividad

A) Ejercicios de operaciones combinadas

Desarrolle los siguientes ejercicios, de manera secuencial y luego verifique con la función print(), si obtuvo el monto generado:

- $3 + 6 * 14 - 20$
- $(100 + 2 * 4 - 5 + 8) * (6 \% 4)$
- $(4 + (30 \% 3) * 4 + 5) * 2$
- $4 * 7 + (2 ** 3) - (40 \% 13)$
- $((15 \% 3) + (34 // 2)) ** 2 - (45 \% 12) - (-20 \% 3)$
- $((10 + (3 * 4) ** 2) - 100) * 3 - ((50 \% 7) * (25 // 7) + 20)$
- $(2 ** 9) * (9 + 3) * (8 // 2) * (10 \% 2) - (50 \% 2)$

B) Ejercicios para convertir expresiones algebraicas en expresiones lineales

Transforme las siguientes expresiones algebraicas en expresiones lineales:

- $5x^2 + y^3$
- $\frac{5x^3y^2}{\sqrt[4]{2c^3}}$
- $\frac{y^3x^2 + 4x^2y + 2y^2x^2 + 8x^2}{xy^4 - 16x}$
- $\frac{a^3b}{2ab^2} - \sqrt{12d^4}$

7. Fuentes consultadas:

- A) <http://runest.ing.puc.cl/basics.html>
- B) <https://j2logo.com/python/tutorial/operadores-en-python/>
- C) <https://edube.org/learn/programming-essentials-in-python-part-1-spanish>
- D) Edison Zavaleta C. (2005). *Fundamentos de Programación*. Perú, Editorial Abaco-Lima.