INTRODUCCION A LA PROGRAMACIÓN PYTHON

En programación, un script es un archivo de código fuente con instrucciones sencillas, que puede ser ejecutado a través de la línea de comandos. Se conoce como scripting a la técnica de programación empleada para crear este tipo de archivos.

Para que un archivo de código fuente sea considerado un script, debe cumplir las siguientes características:

1. Ser ejecutable.
2. Estar escrito en un lenguaje que pueda ser interpretado por el computador.
3. No depender de otros archivos.

LENGUAJE

Es una serie de símbolos que sirven para transmitir uno o más mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como comunicación.

- Lenguaje Máquina
- Lenguaje de Bajo Nivel (Ensamblador)
- Lenguaje de Alto Nivel

LENGUAJE INTERPRETADO

Un script puede ser escrito en cualquier lenguaje interpretado, soportado por el Sistema Operativo, como Python, Perl, GNU Bash, PHP, Ruby, etc.

ALGORITMO

La palabra **algoritmo** se deriva de la traducción al latín de la palabra árabe **alkhowarizmi**, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

Etapas para la solución de un Problema

- Definición del Problema
- Análisis del Problema
- Diseño del Algoritmo
- Codificación
- Prueba y Depuración
- Documentación
- Mantenimiento

PROPIEDADES DE LOS ALGORITMOS

- Corrección: Dar solución al problema.
- Eficiencia: Solución que consuma mínimos recursos (tiempo y espacio).
- Simplicidad: Sencillo de expresar.
- Repetible: Entrega resultados idénticos ante entradas idénticas.
- Finito: Siempre debe terminar.

METODOLOGÍA DISEÑO DESCENDENTE

- Una metodología eficaz en el proceso de diseño se denomina “Divide y Vencerás”.
- La solución del problema plantea la división del problema inicial en subproblemas más sencillos de resolver.
- Este método recibe el nombre de diseño descendente y al proceso de la división en subproblemas cada vez más sencillos se le denomina Refinamiento Sucesivo.
- Esta metodología origina varios nuevos problemas que serán llamados subprogramas o módulos.

ACERCA DE PYTHON

Python, pertenece al grupo de los lenguajes de programación y puede ser clasificado como un lenguaje interpretado, de alto nivel, multiplataforma, de tipado dinámico y multiparadigma.

Para la escritura de código, y a fin de alcanzar un mecanismo estándar en la forma de programar, provee unas reglas de estilo definidas a través de la Python Enhancement Proposal No 8 (PEP 8).

LENGUAJES INFORMATICOS

Es un lenguaje artificial utilizado por computadores, cuyo fin es transmitir información de algo a alguien. Los lenguajes informáticos, pueden clasificarse en:

a) lenguajes de programación (Python, PHP, Pearl, C, etc.);
b) lenguajes de especificación (UML);
c) lenguajes de consulta (SQL);
d) lenguajes de marcas (HTML, XML);
e) lenguajes de transformación (XSLT);
f) protocolos de comunicaciones (HTTP, FTP); entre otros.

LENGUAJE DE PROGRAMACIÓN

Es un lenguaje no natural, diseñado para expresar órdenes e instrucciones precisas, que deben ser llevadas a cabo por un computador. El mismo puede utilizarse para crear programas que controlen el comportamiento físico o lógico de un computador. Está compuesto por una serie de símbolos, reglas sintácticas y semánticas que definen la estructura del lenguaje.

LENGUAJE DE ALTO NIVEL

Son aquellos cuya característica principal, consiste en una estructura sintáctica y semántica legible, acorde a las capacidades cognitivas humanas. A diferencia de los lenguajes de bajo nivel, son independientes de la arquitectura del hardware, motivo por el cual, asumen mayor portabilidad.

LENGUAJES INTERPRETADOS

A diferencia de los compilados, no requieren de compiladores para ser ejecutados, sino de intérpretes. Un intérprete, actúa de manera similar a un compilador, con la salvedad de que ejecuta el programa directamente, sin necesidad de generar previamente un archivo ejecutable. Ejemplo de lenguajes de programación interpretados son Python, PHP, Ruby, Lisp, entre otros.

TIPADO DINAMICO

Un lenguaje de tipado dinámico es aquel cuyas variables, no requieren ser definidas asignando su tipo de datos, sino que éste, se auto asigna en tiempo de ejecución, según el valor declarado.

MULTIPLATAFORMA

Significa que puede ser interpretado en diversos Sistemas Operativos como GNU/Linux, OpenBSD, sistemas privativos, entre otros.

MULTIPARADIGMA

Acepta diferentes paradigmas (técnicas) de programación, tales como la orientación a objetos, la programación imperativa y funcional.

CODIGO FUENTE

Es un conjunto de instrucciones y órdenes lógicas, compuestos de algoritmos que se encuentran escritos en un determinado lenguaje de programación, las cuales deben ser interpretadas o compiladas, para permitir la ejecución de un programa informático.

ELEMENTOS DEL LENGUAJE PYTHON

Python, al igual que todo lenguaje de programación, se compone de una serie de elementos y estructuras que componen su sintaxis. A continuación, se abarcarán los principales elementos.

VARIABLES

Una variable es un espacio para almacenar datos modificables, en la memoria del computador.

En Python, una variable se define con la sintaxis:

```
nombre_de_la_variable = valor_de_la_variable
```

Cada variable, tiene un nombre y un valor. Este último, define el tipo de datos de la variable.

Existe un tipo de espacio de almacenamiento denominado constante y se utiliza para definir valores fijos, que no requieran ser modificados. En Python, el concepto de constante es simbólico ya que todo espacio es variable.

Recomendación

Utilizar nombres descriptivos y en minúsculas. Para nombres compuestos, separar las palabras por guiones bajos.
Antes y después del signo =, debe haber uno (y solo un) espacio en blanco
Ejemplo

```
mi_variable = 12
```

Recomendación

Para constantes
Utilizar nombres descriptivos y en mayúsculas separando palabras por guiones bajos.
Ejemplo:

```
MI_CONSTANTE = 12
```

ENTRADA Y SALIDA

Para imprimir un valor en pantalla se utiliza la función **`print()`**:

```
mi_variable = 15
print(mi_variable)
```

Para obtener datos solicitados al usuario se utiliza la función **`input()`.**

```
mi_variable = input("Ingresar un valor: ")
```

TIPOS DE DATOS

Una variable puede contener valores de diversos tipos. Estos tipos se listas a continuación.

Cadena de texto (string):

```
mi_cadena = "Hola Mundo!"
otra_cadena = 'Hola Mundo!'

mi_cadena_multilinea = """
Esta es una cadena
de varias lineas
"""
```

Número entero:

```
edad = 35
```

Número real:

```
precio = 35.05
```

Booleano (verdadero / Falso):

```
verdadero = True
```

```
falso = False
```

Existen además, otros tipos de datos más complejos, que se abarcarán más adelante.

OPERADORES ARITMÉTICOS

Un operador aritmético es aquel que permite realizar operaciones aritméticas sobre las variables.

Símbolo	Significado	Ejemplo	Resultado
+	Suma	a = 10 + 5	a es 15
-	Resta	a = 12 - 7	a es 5
-	Negación	a = -5	a es -5
*****	Multiplicación	a = 7 * 5	a es 35
****	Potenciación	a = 2 ** 3	a es 8
/	División	a = 12.5 / 2	a es 6.25
//	División entera	a = 12.5 // 2	a es 6
%	Módulo	a = 27 % 4	a es 3

Recomendación
Siempre colocar un espacio en blanco, antes y después de un operador.

Un ejemplo sencillo con variables y operadores aritméticos:

```
monto_bruto = 175
tasa_interes = 12.5
monto_interes = monto_bruto * tasa_interes / 100
tasa_bonificacion = 5.0
```

```
importe_bonificacion = monto_bruto * tasa_bonificacion / 100
monto_neto = (monto_bruto - importe_bonificacion) + monto_interes
```

COMENTARIOS

En un archivo de código fuente, un comentario es un texto explicativo que no es procesado por el intérprete y solo tiene una utilidad humana (no informática). Los comentarios pueden ser de una o varias líneas:

```
# Esto es un comentario de una sola línea
mi_variable = 15
"""Y este es un comentario
de varias líneas"""

mi_variable = 15
mi_variable = 15 # Este es un comentario en línea
```

En los comentarios, pueden incluirse palabras que representen acciones:

```
# TODO esto es algo por hacer
# FIXME esto es algo que debe corregirse (un fallo)
# XXX esto es algo que debe corregirse (error crítico)
```

Recomendación
Comentarios en la misma línea del código deben separarse con dos espacios en blanco. Luego del símbolo # debe ir un solo espacio en blanco. Correcto: a = 15 # Edad de María

Ejemplo:

Suponga que un individuo quiere invertir su capital en un banco y desea saber cuánto dinero ganará después de un mes si el banco paga a razón de 2% mensual.

TIPOS DE DATOS COMPLEJOS

Python, posee además de los tipos ya vistos, 3 tipos más complejos, que admiten una colección de datos. Estos tipos son:

- Tuplas
- Listas
- Diccionarios

Estos tres tipos, pueden almacenar colecciones de datos de diversos tipos y se diferencian por su sintaxis y por la forma en la cual los datos pueden ser manipulados.

TUPLAS

Una tupla es una variable que permite almacenar varios datos inmutables (no pueden ser modificados una vez creados), y estos datos pueden ser de tipos diferentes:

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)
```

Se puede acceder a cada uno de los datos mediante su índice correspondiente. El índice se corresponde con la posición del elemento en la colección, siendo 0 (cero), el índice del primer elemento:

```
mi_tupla[1]  # Salida: 15
```

También se puede acceder a una porción de la tupla, indicando (opcionalmente) desde el índice de inicio hasta el índice de fin:

```
mi_tupla[1:4]  # Devuelve: (15, 2.8, 'otro dato')
mi_tupla[3:]   # Devuelve: ('otro dato', 25)
mi_tupla[:2]   # Devuelve: ('cadena de texto', 15)
```

Otra forma de acceder a la tupla de forma inversa (de atrás hacia adelante), es colocando un índice negativo:

```
mi_tupla[-1]  # Salida: 25
mi_tupla[-2]  # Salida: otro dato
```

LISTAS

Una lista es similar a una tupla en todos los aspectos. La diferencia radica en que los elementos de una lista sí pueden ser modificados:

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]
```

A las listas se accede igual que a las tuplas, por su número de índice:

```
mi_lista[1]    # Salida: 15
mi_lista[1:4]  # Devuelve: [15, 2.8, 'otro dato']
mi_lista[-2]   # Salida: otro dato
```

Las lista no son inmutables: permiten modificar los datos una vez creados:

```
mi_lista[2] = 3.8  # el tercer elemento ahora es 3.8
```

Al poder ser modificadas, las listas, a diferencia de las tuplas, permiten agregar nuevos valores:

```
mi_lista.append('Nuevo Dato')
```

DICCIONARIOS

Los diccionarios, al igual que las tuplas y las listas, son colecciones. La diferencia con estos, es que mientras que los elementos de las listas y las tuplas se asocian a un número de índice (o posición), los valores de un diccionario se asocian a un nombre de clave:

```
mi_diccionario = {
    'clave_1': valor_1,
    'clave_2': valor_2,
    'clave_7': valor_7
}

mi_diccionario['clave_2']  # Salida: valor_2
```

Un diccionario permite eliminar cualquier entrada:

```
del(mi_diccionario['clave_2'])
```

Al igual que las listas, el diccionario permite modificar los valores:

```
mi_diccionario['clave_1'] = 'Nuevo Valor'
```

Y también es posible agregar nuevos elementos, asignando valores a nuevas claves:

```
mi_diccionario['nueva_clave'] = 'nuevo elemento'
```

ESTRUCTURAS DE CONTROL DE FLUJO

Una estructura de control es un bloque de código fuente que permite agrupar instrucciones de forma controlada. A continuación se describirán dos tipos de estructuras de control:

1. Estructuras de control condicionales: controlan el flujo de los datos a partir de condiciones.

2. Estructuras de control iterativas: controlan el flujo de los datos, ejecutando una misma acción de forma repetida.

IDENTACIÓN

En Python, las estructuras de control se delimitan sobre la base de bloques de código identados. En un lenguaje informático, se llama identación al sangrado del código fuente.

No todos los lenguajes de programación, necesitan de una identación, aunque sí se estila implementarla, a fin de otorgar mayor legibilidad al código fuente. En el caso de Python, la identación es obligatoria, ya que de ella, dependerá su estructura y la forma de controlar la información.

Recomendación

Una identación de 4 (cuatro) espacios en blanco, indicará que las instrucciones identadas, forman parte de una misma estructura de control.

Una estructura de control, entonces, se define de la siguiente forma:

```
inicio de la estructura de control:
    expresiones
```

ESTRUCTURAS DE CONTROL DE FLUJO CONDICIONALES

Las estructuras de control condicionales son aquellas que permiten evaluar si una o más condiciones se cumplen, para decir qué acción ejecutar.

Las condiciones se evalúan como verdaderas o falsas. O la condición se cumple (la condición es verdadera), o la condición no se cumple (la condición es falsa).

Las estructuras de control de flujo condicionales, se definen mediante el uso de tres palabras claves reservadas, del lenguaje: `if (si), elif (sino, si) y else (sino).`

```
if semaforo == verde:
    cruzar()
else:
    esperar()
```

Símbolo	Significado	Ejemplo	Resultado
==	Igual que	`5 == 7`	Falso
!=	Distinto que	`rojo != verde`	Verdadero
<	Menor que	`8 < 12`	Verdadero
>	Mayor que	`7 > 12`	Falso
<=	Menor o igual que	`12 <= 12`	Verdadero
>=	Mayor o igual que	`4 >= 5`	Falso

Para evaluar más de una condición simultáneamente, se utilizan operadores lógicos:

Operador	Ejemplo	Evaluación	Resultado
and (y)	`5 == 7 and 7 < 12`	Falso y Falso	Falso
	`9 < 12 and 12 > 7`	Verdadero y Verdadero	Verdadero
	`9 < 12 and 12 > 15`	Verdadero y Falso	Falso
or (o)	`12 == 12 or 15 < 7`	Verdadero o Falso	Verdadero
	`7 > 5 or 9 > 3`	Falso o Falso	Falso

Ejemplo:

Si gasto hasta $100, pago con dinero en efectivo. Sino, si gasto más de $100 pero menos de $300, pago con tarjeta de débito. Sino, pago con tarjeta de crédito.

```
if gasto <= 100:
    pagar_en_efectivo()
elif gasto > 100 and gasto < 300:
    pagar_con_debito()
else:
    pagar_con_credito()
```

ESTRUCTURAS DE CONTROL ITERATIVAS

Las estructuras de control iterativas (también llamadas cíclicas o bucles), permiten ejecutar un mismo código, de forma repetida, mientras se cumpla una determinada condición.

Python dispone dos estructuras de control iterativas:

- El ciclo while
- El ciclo for

CICLO WHILE

Este ciclo se encarga de ejecutar una misma acción mientras que una determinada condición se cumpla:

Ejemplo:

Mientras que el año sea menor o igual a 2022, imprimir la frase “Informes de <año>”.

Código Python que da respuesta al problema enunciado.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
year = 2011
while year <= 2022:
     print ("Informes de”, year)
     year = year + 1
```

Esto mostrara en pantalla lo siguiente:

```
Informes de 2011
Informes de 2012
Informes de 2013
Informes de 2014
Informes de 2015
Informes de 2016
Informes de 2017
Informes de 2018
Informes de 2019
Informes de 2020
Informes de 2021
Informes de 2022
```

En cada iteración se incrementa en 1 (uno) el valor de la variable year que condiciona el ciclo. Si no se incrementase esta variable, su valor siempre sería igual a 2001 y por lo tanto, el ciclo se ejecutaría de forma infinita, ya que la condición year <= 2012 siempre se estaría cumpliendo.

Romper un ciclo

Cuando el valor que condiciona la iteración no puede incrementarse, puede romperse el ciclo utilizando la instrucción break precedida de un condicional:

```
while True:
    nombre = input("Indique su nombre: ")
    if nombre:
          break
```

El ciclo anterior, incluye una condición anidado que verifica si la variable nombre es verdadera (solo será verdadera si el usuario aporta un dato cuando le es solicitado en pantalla). Si es verdadera, el bucle se rompe (break). Sino, seguirá ejecutándose hasta que el usuario ingrese un dato cuando le es solicitado.

Ciclo FOR

El ciclo for de Python facilita la iteración sobre los elementos de una tupla o lista. El ciclo for siempre se utilizará sobre una lista o tupla, de forma tal que en cada iteración, se puedan ejecutar las misma acciones sobre cada uno de los elementos de la lista o tupla.

```
#!/usr/bin/env python3
mi_lista = ['Juan', 'Antonio', 'Pedro', 'Ana']
for elemento in mi_lista:
     print(elemento)
```

Otra forma de iterar con el bucle for, puede emular a while:

Ejemplo:

Mientras que el año sea menor o igual a 2022, imprimir la frase “Informes de <año>”.

```
for year in range(2010, 2023):
    print ("Informes de", year)
```

Ejemplo de programación.

Suponga que un individuo quiere invertir su capital en un banco y desea saber cuánto dinero ganará después de un mes si el banco paga a razón de 2% mensual.

Definición del problema.

Como ya se indicó, la primera etapa es comprender el problema, ser capaz de definirlo con sus propias palabras.

Análisis del problema.

La siguiente etapa, es el análisis del mismo poniendo el énfasis en “***comprender***” los datos de entrada, caracterizarlos (string, enteros, reales, etc.), que nombre les daré en el programa, que relación tienen entre ellos o como pueden ayudar en la solución del problema. Se continua con el análisis de los datos de salida (que es lo que están pidiendo calcular, como debo mostrar el resultado, como llego a una respuesta al problema con ayuda de los datos de entrada, etc.).

Conviene en esta etapa realizar el siguiente bosquejo:

Datos de entrada

Capital a invertir (cap_invertir), es un numero entero.

Interés pagado por el banco = 2% mensual, es un número real (float).

Datos de salida

Ganancia obtenida en un mes (ganancia), es un numero float.

Diseño del algoritmo.

En este punto se debe establecer que cálculos se deben realizar para lograr dar respuesta al problema, es una descripción de como usted visualiza la solución del problema, esta visualización puede ser por medio de un gráfico, una glosa (un texto), un diagrama, etc., que permita explicitar los pasos a seguir para dar respuesta al problema.

Un ejemplo en glosa sería:

En primer lugar se debe solicitar que se ingrese un valor numérico entero que corresponderá al capital que se desea invertir. A continuación, como ya se conoce el interés que paga el banco (es un porcentaje), se procede a calcular dicho porcentaje sobre lo ingresado como capital a invertir. (ver como se calcula un determinado porcentaje). Este valor calculado es la ganancia que debe ser reportada al como solución al problema.

Este algoritmo (efectivamente es el algoritmo), debe ser depurado y refinado hasta llegar a instrucciones en algún lenguaje de programación. Por ejemplo como se ingresan datos en este lenguaje, como se asignan valores a variables como se hacen las operaciones aritméticas, etc.

Codificación.

… En primer lugar se debe solicitar que se ingrese un valor numérico entero que corresponderá al capital que se desea invertir…

Corresponde a:

```
cap_invertir = input("Ingrese capital a invertir: ")
```

En este punto conviene recordar que la instrucción input solo captura string y no valores enteros, por lo que se debe estudiar como transformar un string en número entero en Python (puesto que es el lenguaje en el que estamos codificando).

```
cap_invertir = int(cap_invertir)
```

Así es como logramos convertir desde string a número entero.

…A continuación, como ya se conoce el interés que paga el banco (es un porcentaje), se procede a calcular dicho porcentaje sobre lo ingresado como capital a invertir. (ver como se calcula un determinado porcentaje)…

```
ganancia = cap_invertir * 0.02
```

Este cálculo nos entrega la ganancia que se obtendrá por el capital invertido.

… Este valor calculado es la ganancia que debe ser reportada al como solución al problema….

```
print ("La ganancia obtenida en el mes es: ", ganancia)
```

El resultado final es el siguiente programa:

```
cap_invertir = input("Ingrese capital a invertir: ")
cap_invertir = int(cap_invertir)
ganancia = cap_invertir * 0.02
print ("La ganancia obtenida en el mes es: ", ganancia)
```

Lo ejemplificando es la manera de enfrentar los problemas en computación.

En los siguientes ejercicios ponga en práctica lo indica mas arriba.

Ejercicio de programación

1) Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.
2) Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.
3) Un profesor conoce la cantidad de hombres y mujeres del curso de Algoritmos, desea saber qué porcentaje de hombres y que porcentaje de mujeres hay en su grupo de estudiantes.
4) Calcular el nuevo sueldo de un empleado si obtuvo un incremento del 25% sobre su sueldo anterior.
5) Tres personas deciden invertir su dinero para fundar una empresa. Cada una de ellas invierte una cantidad distinta. Obtener el porcentaje que cada quien invierte con respecto a la cantidad total invertida.
6) Un alumno desea saber cuál será su nota final en el curso de Introducción a la programación. Dicha nota se compone de los siguientes porcentajes: 55% del promedio de sus tres notas parciales, 30% de la nota del examen final y 15% de la nota de un trabajo final.
7) En un hospital existen tres áreas: Ginecología, Pediatría, Traumatología. El presupuesto anual del hospital se reparte conforme a la siguiente manera: Ginecología un 50%, Traumatología un 20% y Pediatría un 30%. Obtener la cantidad de dinero que recibirá cada área, para cualquier monto presupuestado.
8) Un estudiante desea saber cuál será su promedio general en los tres cursos más difíciles que cursa y cuál será el promedio que obtendrá en cada una de ellas. Estas asignaturas se evalúan como se muestra a continuación:

   La calificación de Matemáticas se obtiene de la sig. manera:

Examen 90%

Promedio de tareas 10%

En esta asignatura se pidió un total de tres tareas.

La calificación de Física se obtiene de la sig. manera:

Examen 80%

Promedio de tareas 20%

En esta asignatura se pidió un total de dos tareas.

La calificación de Programación se obtiene de la sig. manera:

Examen 85%

Promedio de tareas 15%

En esta asignatura se pidió un total de tres tareas.

La nota promedio general es ponderada con los porcentajes siguientes: Matemática 30%, Física 30% y Computación 40%.

FUNCIONES

Una función es una forma de agrupar expresiones y algoritmos de tal forma que estos queden contenidos dentro de «una cápsula», que solo pueda ejecutarse cuando el programador la invoque. Una vez que una función es definida, puede ser invocada tantas veces como sea necesario.

Funciones incorporadas

Una función puede ser provista por el lenguaje o definida por el usuario. A las funciones provistas por el lenguaje se las denomina «funciones incorporadas» y en inglés se conocen como «Built-in functions».

FUNCIONES DEFINIDAS POR EL USUARIO

En Python la definición de funciones se realiza mediante la instrucción **`def`** más un nombre de función descriptivo para el cuál aplican las mismas reglas que para el nombre de las variables seguido de paréntesis de apertura y cierre.
Como toda estructura de control en Python, la definición de la función finaliza con dos puntos (:) y el algoritmo que la compone, irá identado con 4 espacios en blanco:

```
def mi_funcion():
     # aquí debe ir código identado
```

Una función no es ejecutada hasta tanto no sea invocada. Para invocar una función, simplemente se la llama por su nombre:

```
def mi_funcion():
  print ("Hola Mundo")

funcion()
```

Las funciones pueden retornar datos:

```
def funcion():
    return "Hola Mundo"
```

Y el valor de retorno de una función puede almacenarse en una variable:

```
    frase = funcion()
```

Imprimirse:

```
    print (funcion())
```

O ignorarse:

```
    funcion()
```

SOBRE LOS PARÁMETROS

Un parámetro es un valor que la función espera recibir cuando sea llamada (invocada). Una función puede esperar uno o más parámetros (que son definidos entre los paréntesis y separados por una coma) o ninguno.

```
def mi_funcion(param1, param2):
    pass
```

Observación La instrucción **pass** se utiliza para completar una estructura de control que no realiza ninguna acción.

Los parámetros que una función espera, serán utilizados por ésta, dentro de su código estos parámetros serán variables locales que solo son accesibles dentro de la función:

```
def calcular_neto(bruto, valor_iva):
    iva = bruto * float(valor_iva) / 100
    neto = bruto + iva
    return neto
```

Si se quisiera acceder a esas variables locales fuera del ámbito de la función, se obtendría un error.

Observación

Al llamar a una función se le deben pasar sus argumentos en el mismo orden en el que los espera.

SOBRE LA FINALIDAD DE LAS FUNCIONES

Una función puede tener cualquier tipo de algoritmo y cualquier cantidad de instrucciones. Sin embargo, una buena práctica indica que la finalidad de una función, debe ser realizar una única acción.