Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

Funciones

Simples, funciones desde otra función, funciones anidadas

Hechos QUE CONECTAN

Contenido

- Funciones
  - Llamado de funciones
- Funciones propias de Python
- Funciones creadas por nosotros
  - Sentencia **def**
- Funciones desde otra función
- Funciones anidadas

Contenido

Funciones

- Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea, este bloque puede ser llamados cuando se necesite.

Contenido

Funciones

- El uso de funciones es un componente muy importante del paradigma de la programación llamada estructurada, y tiene varias ventajas:

Contenido

Funciones

- • Modularización: permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.
- • Reutilización: permite reutilizar una misma función en distintos programas.

Contenido

Funciones

- Python dispone de una serie de funciones integradas al lenguaje, y también permite crear funciones definidas por el usuario para ser usadas en su propios programas.

Contenido

Llamado de funciones

En el contexto de la programación, una función es una secuencia de declaraciones con nombre que realiza un cálculo. Cuando se define una función, se especifica el nombre y la secuencia de declaraciones. Más tarde, puede "llamar" la función por su nombre. Ya hemos visto un ejemplo de llamada de una función:

```
print("HOLA MUNDO")
```

Contenido

Algunas funciones

print() = Imprime por consola
int() = define un valor entero
type() = imprime el tipo de valor
round() = redondea numero con decimal o establece numero máximo de decimales
max() = selecciona mayor valor de lista
min() = selecciona menor valor de lista
help() = ayuda sobre palabra reservada

Algunas funciones

sum() = suma un rango de valores
range() = establece un rango
return() = devuelve argumentos

Contenido

Funciones propias de Python

Python proporciona una serie de importantes funciones incorporadas que podemos utilizar sin necesidad de proporcionar la definición de la función. Los creadores de Python escribieron un conjunto de funciones para resolver problemas comunes y las incluyeron en Python para que las usáramos.

A continuación veremos unos ejemplos:

Contenido

Max y Min

Las funciones max y min nos dan los valores más grandes y más pequeños de un arreglo, respectivamente. Este arreglo puede ser una lista, un vecto e incluso un conjunto de variables:

```
numeros = [25, 26, 25, 24]

maximo = max(numeros)
print(maximo)
```

```
minimo = min(numeros)
print(minimo)

suma = sum(numeros)
print(suma)
```

Contenido

Help

La funcion Help llama al sistema de ayuda integrado de Python.

help(print)

salida?

help(max)

salida?

help(min)

salida?

help(type)

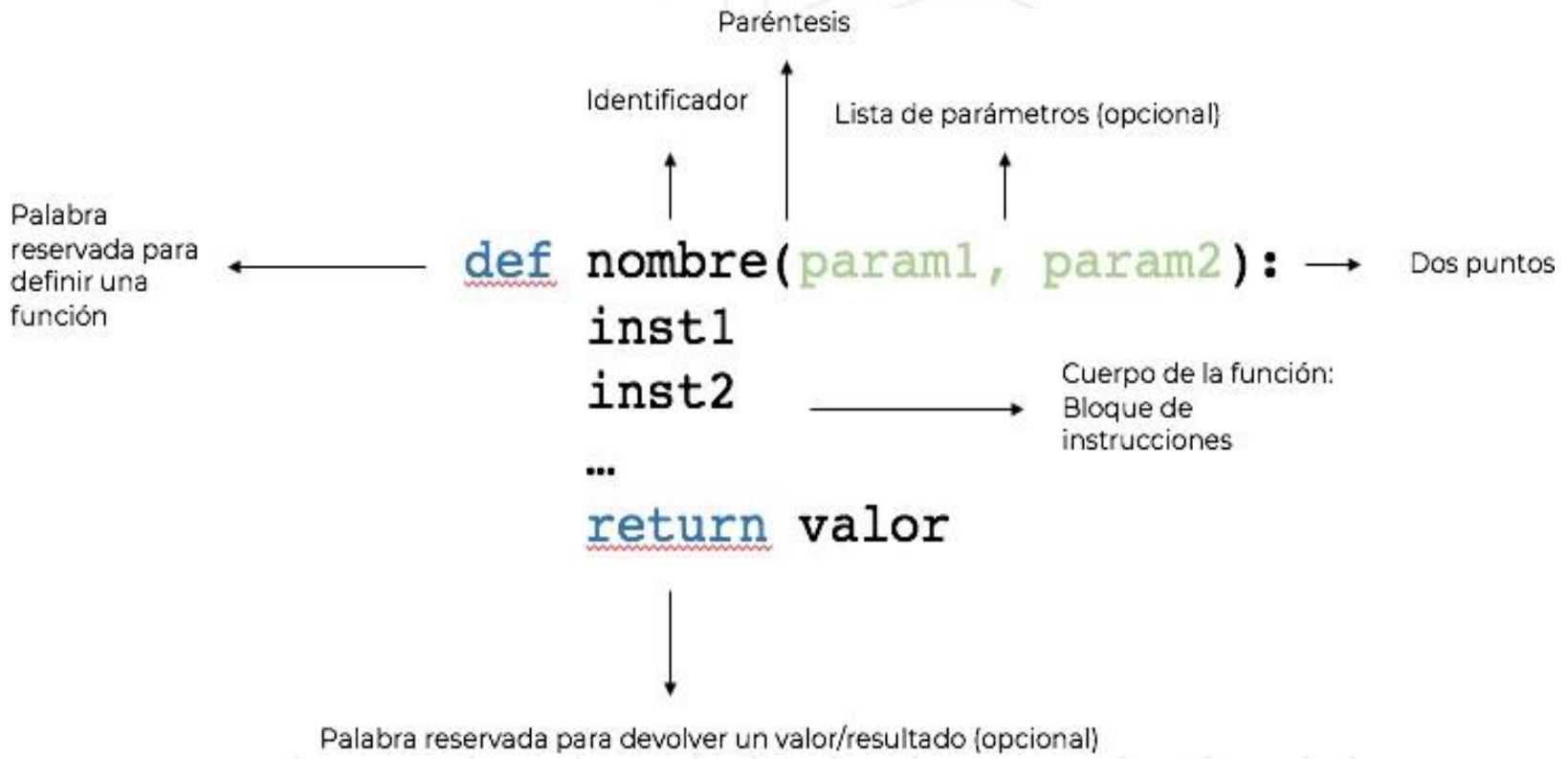
salida?

Contenido

Funciones propias

Hasta ahora, sólo hemos usado las funciones que vienen con Python, pero también es posible añadir y crear nuevas funciones. Una definición de función especifica el nombre de una nueva función y la secuencia de declaraciones que se ejecutan cuando se llama la función. Una vez que definimos una función, podemos reutilizar la función una y otra vez a lo largo de nuestro programa.

Contenido


Paréntesis
Identificador
Lista de parámetros (opcional)
Palabra reservada para definir una función
def nombre(param1, param2):
Dos puntos
inst1
inst2
Cuerpo de la función: Bloque de instrucciones
...
return valor
Palabra reservada para devolver un valor/resultado (opcional)

Contenido

```
#Crear una función:
def func(parámetro, parámetro):
    #código de la función <-- Identación
    return

#Llamar la función:
func(argumento, argumento)

#Almacenar el retorno de la función (si tiene uno):
variable = func(argumento, argumento)
```

Al incluir un **“return”** le estamos diciendo a python que retorne inmediatamente el valor de resultado de la función y use la siguiente expresión como un valor de retorno”

Contenido

Sentencia def

La sentencia def es una definición de función usada para crear objetos funciones definidas por el usuario.
Una definición de función es una sentencia ejecutable. Su ejecución enlaza el nombre de la función en el namespace local actual a un objecto función (un envoltorio alrededor del código ejecutable para la función). Este objeto función contiene una referencia al namespace local global como el namespace global para ser usado cuando la función es llamada.
La definición de función no ejecuta el cuerpo de la función; esto es ejecutado solamente cuando la función es llamada.

Contenido

La sintaxis para una definición de función en Python es:

Tal como cualquier otro lenguaje de programación, los operadores de sumas, restas, multiplicaciones y divisiones pueden ser usadas con números.

Contenido

```
def NOMBRE(LISTA_DE_PARAMETROS):
    """DOCSTRING_DE_FUNCION"""
    SENTENCIAS
    RETURN [EXPRESION]
```

A continuación se detallan el significado de pseudo código fuente anterior:

- **NOMBRE**, es el nombre de la función.
- **LISTA_DE_PARAMETROS**, es la lista de parámetros que puede recibir una función.
- **DOCSTRING_DE_FUNCION**, es la cadena de caracteres usada para documentar la función.
- **SENTENCIAS**, es el bloque de sentencias en código fuente Python que realizar cierta operación dada.
- **RETURN**, es la sentencia return en código Python.
- **EXPRESION**, es la expresión o variable que devuelve la sentencia return.

Contenido

Vamos a crear nuestra primera función

```
def imprime_Cosas():
    print("La clase esta genial")
    print('Python es lo máximo')

imprime_Cosas()

def repetir_funciones():
    print("\n")
    imprime_Cosas()
    imprime_Cosas()

repetir_funciones()
```

Contenido

Probemos este ejercicio

```
def repetir_funciones():
    print("\n")
    imprime_Cosas()
    imprime_Cosas()

repetir_funciones()

def imprime_Cosas():
    print("La clase esta genial")
    print('Python es lo máximo')

imprime_Cosas()
```

Contenido

Vamos a crear nuestra primera función

```
def sumarDosnumeros():
    num1 = float(input("Ingrese el numero 1: "))
    num2 = float(input("Ingrese el numero 2: "))
    print("la suma de", int(num1), " + ", int(num2), "  es igual a: ", int(num1 + num2))

sumarDosnumeros()
```

Contenido

Vamos a crear nuestra primera función

```
def raizCuadrada():
    valor = float(input("Por favor, ntroduzca un numero a calcaluar su raiz cuadrada: "))
    raiz = valor ** 0.5
    return print("La raiz cuadrada de : ", valor, " es ", valor ** 0.5)

raizCuadrada()
```

Contenido

Flujo de ejecución

- Para asegurarse de que una función está definida antes de su primer uso, hay que conocer el orden en el que se ejecutan las declaraciones, lo que se denomina flujo de ejecución.
- La ejecución siempre comienza en la primera sentencia del programa. Las sentencias se ejecutan una a una, en orden de arriba a abajo. Las definiciones de la función no alteran el flujo de ejecución del programa, pero recuerda que las sentencias dentro de la función no se ejecutan hasta que la función es llamada.Una llamada a la función es como un desvío en el flujo de ejecución.
- En lugar de ir a la siguiente declaración, el flujo salta al cuerpo de la función, ejecuta todas las declaraciones allí, y luego vuelve a retomar donde lo dejó. Eso suena bastante simple, hasta que recuerdas que una función puede llamar a otra. Mientras que en medio de una función, el programa podría tener que ejecutar las declaraciones en otra función. Pero mientras ejecuta esa nueva función, el programa podría tener que ejecutar otra función.
- Afortunadamente, Python es bueno para llevar la cuenta de dónde está, así que cada vez que una función se completa, el programa continúa donde lo dejó en la función que la llamó. Cuando lees un programa, no siempre quieres leer de arriba a abajo. A veces tiene más sentido si sigues el flujo de la ejecución.

Contenido

Argumentos y parámetros

Por posición

Cuando envía argumentos a una función, estos se reciben por orden en los parámetros definidos. Se dice por tanto que son argumentos por posición:

```
def suma(a, b):
    return a + b

print(suma(30, 10))
```

Contenido

Argumentos y parámetros

Por nombre

Sin embargo es posible evadir el orden de los parámetros si indica durante la llamada que valor tiene cada parámetro a partir de su nombre:

```
def suma(a, b):
      return a + b
B = 30
A = 10
print(suma(a, b))
```

Contenido

Sentencia *return*

Por nombre

Las funciones pueden comunicarse con el exterior de las mismas, al proceso principal del programa usando la sentencia return. El proceso de comunicación con el exterior se hace devolviendo valores. A continuación, un ejemplo de función usando return:

Contenido

Sentencia *return*

Ejemplo

```
def otra_suma(numero1, numero2):
    print(numero1 + numero2)
    print("\n")

resultado = otra_suma(5,6)
print(resultado)
```

```
def otra_suma(numero1,numero2):
    print(numero1 + numero2)
    print("\n")
    return numero1 + numero2

otra_suma(5, 6)
```

Contenido

Sobre los parámetros

Un parámetro es un valor que la función espera recibir cuando sea llamada (invocada), a fin de ejecutar acciones en base al mismo. Una función puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.

Contenido

```
def mi_funcion(nombre, apellido):
    # algoritmo
```

Los parámetros, se indican entre los paréntesis, a modo de variables, a fin de poder utilizarlos como tales, dentro de la misma función.
Los parámetros que una función espera, serán utilizados por ésta, dentro de su algoritmo, a modo de variables de ámbito local. Es decir, que los parámetros serán variables locales, a las cuáles solo la función podrá acceder:

```
def mi_funcion(nombre, apellido):
    miNombre = nombre + apellido
    return(miNombre)

print(mi_funcion("Luis ", "Molero"))
```

Contenido

Parámetros por omisión

En Python, también es posible, asignar valores por defecto a los parámetros de las funciones. Esto significa, que la función podrá ser llamada con menos argumentos de los que espera:

```
def saludar(nombre, mensaje='Hola'):
    print(mensaje, nombre)

saludar('Pepe Grillo')
```

Contenido

Funciones dentro de otra función

Vamos a crear nuestra primera función

```
def imprime_Cosas():
    print("La clase esta genial")
    print('Python es lo máximo')

imprime_Cosas()

def repetir_funciones():
    imprime_Cosas()
    imprime_Cosas()

repetir_funciones()
```

Contenido

Vamos a crear nuestra primera función

```
def mensaje():
    print("Ingrese por favor un valor")

def sumarDosnumeros():
    mensaje()
    num1 = float(input())
    mensaje()
    num2 = float(input())
    return print("la suma de ",num1, " + ", num2, " es igual a: ", num1 + num2)

sumarDosnumeros()
```

Contenido

Vamos a crear nuestra primera función

```
def mensaje():
    print("Por favor, Introduzca un numero a calcaluar su raiz cuadrada: ")

def raizCuadrada():
    mensaje()
    valor = float(input())
    raiz = valor ** 0.5
    return print("La raiz cuadrada de : ", valor, " es ", valor ** 0.5)

raizCuadrada()
```

Contenido

Funciones anidadas

En Python, cualquier función escrita puede ser llamada por otra función. Tenga en cuenta que esta podría ser la forma más elegante de dividir un problema en trozos de pequeños problemas. Ahora, aprenderemos cómo podemos llamar a una función definida desde otra función con la ayuda de varios ejemplos.

Contenido

Vamos a crear nuestra primera función

A continuación, veremos un ejemplo muy sencillo acerca de las funciones anidadas

```
def primeraFuncion(): # función externa
    print ("\n \"Hola desde la función externa\" \n ")
    def segundaFuncion(): # función interna
        print ("\n \"Hola desde la función interna\" \n")

    segundaFuncion()

primeraFuncion()
```

Contenido

Vamos a crear nuestra primera función

En el siguente ejemplo, veremos como una función interna puede acceder a variables accesibles en la función externa.

```
def primerNumero(x):
    def segundoNumero(y):
        return x * y
    return segundoNumero

resultado = primerNumero(2)

print(resultado(5))
```

Contenido

Vamos a crear nuestra primera función

En el siguente ejemplo, veremos como cambiar las variables de la función externa desde dentro de la función interna

```
def primeraFuncion():
    x = 2
    def segundaFuncion(a):
        x = 6
        print(a + x)
    print(x)
    segundaFuncion(3)
primeraFuncion()
```