Esta clase va a ser

- grabada

Clase 08. PYTHON

Funciones I

Temario

07

Controladores de Flujo II

- ✓ Sentencias iterativas
- ✓ While
- ✓ For

08

Funciones I

- ✓ Funciones
- ✓ Retornando valores
- ✓ Enviando valores

09

Funciones II

- ✓ Argumentos y parámetros
- ✓ Funciones recursivas
- ✓ Funciones integradas

CODERHOUSE

Objetivos de la clase

- **Crear** funciones
- **Retornar** valores
- **Enviar** valores

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Funciones

Funciones

Cuando creamos nuestros propios programas nos damos cuenta de que muchas de las tareas que implementamos se repiten o se presentan de forma similar pero con algunos cambios.

Entonces aparece la necesidad de agrupar este código repetido o similar, a las agrupaciones de código se les denomina **funciones** las cuales se pueden ejecutar múltiples veces gracias a un nombre único que las identifica.

Funciones

Para comunicarse con nuestro proceso principal, las funciones pueden recibir y devolver datos manipulados. Un ejemplo de una función que conocemos es len() que nos permite saber la cantidad de elementos de una colección

Recordemos que a esta función hay que pasarle el elemento del cual queremos saber la longitud y devuelve un valor entero con la longitud, a este valor se le denomina valor de retorno.

```
>>> len("Hola")
4
```

DEF

¿De qué trata?

La sentencia def sirve para crear funciones definidas por el usuario. Una definición de función es una sentencia ejecutable.

Sintaxis para una definición de función

```
>>> def NOMBRE(PARÁMETROS):
        SENTENCIAS
        RETURN [EXPRESIÓN]
```

- ✓ **Nombre:** Es el nombre de la función
- ✓ **Parámetro**: Como vimos en la clase 8 hay scripts con argumentos, en las funciones, cuando recibe argumentos, se les denominan parámetros.
- ✓ **Sentencias**: Es el bloque del código.
- ✓ **Return:** Es una sentencia de Python, le indica a la función qué devolver cuando llamamos a la función
- ✓ **Expresión:** Es lo que devuelve la sentencia return.

Definir funciones básicas

```
def saludar():
    print('Estoy saludando desde la función')
```

La llamamos usando:

```
saludar()
```

Estoy saludando desde la función

Definir funciones más avanzadas

```
[6]  def saludar_con_nombre(nombre):
         saludando = print("Hola {}! ¿Cómo estás?".format(nombre)")
         return saludando
```

```
saludar_con_nombre("Juan")
```

```
Hola {}! ¿Cómo estás?".format(nombre)
```

Prueba el código para ver qué pasa 🙂

Recomendaciones

- Utilizar minúsculas
- Las palabras se separan con guiones bajos _
- Utilizar nombres autoexplicativos
- No usar nombres que no definan lo que hace la función (ejemplo letras simples o palabras sin sentido con lo que haga la función)

Variables y funciones

```
def test():
    variable_test = 10
    print(variable_test)

print(variable_test)
```

```
NameError: name 'variable_test' is not defined
```

Ten en cuenta que las variables creadas en una función **no existen** fuera de la misma, de decir son variables locales.

Para pensar

Si las variables creadas en una función, solo existen dentro de esa función ¿Cómo explicarías esto?

```
>>> variable_test = 10
>>> def test():
        print(variable_test)
>>> test()
```

Contesta mediante el chat de Zoom

CODERHOUSE

Variables y funciones

Sin embargo, hay que tener cuidado con las variables fuera de las funciones al usarlas en una función, ya que no puede llegar a funcionar como queremos:

El print le da prioridad a la variable dentro de la función antes que a la de afuera.

```
variable_test = 10
def test():
   variable_test = 155
   print(variable_test)
test()

>>>155
```

Retornando valores

Return

Las funciones pueden comunicarse con el exterior de las mismas, al proceso principal del programa usando la sentencia return. La comunicación con el exterior se hace devolviendo valores.

A continuación, un ejemplo de función usando return:

```
def saludar_con_nombre(nombre):
    saludando = print('Hola {}! ¿Cómo
estás?'.format(nombre))
    return saludando

saludar_con_nombre("Juan")

>>>Hola Juan ¿Cómo estás?
```

Nota: Por defecto, las funciones retorna el valor None.

Return

Sin embargo, hay que tener en cuenta que la función termina al devolver un valor, es decir, lo que escribamos después no se ejecutará:

```
def saludar_con_nombre(nombre):
    saludando = print('Hola {}! ¿Cómo
estás?'.format(nombre))
    return saludando
    print("Hola Mundo")
```

¡Es similar a un break!

Return

Algo interesante que pasa si devolvemos una colección es que podemos utilizarla directamente desde la función y hacer uso de las funciones internas de las colecciones:

Sin embargo, cada vez que hagamos un print a una función la estaremos llamando, por lo que lo ideal es asignarlo a una variable y trabajarlo desde ahí:

```
def lista():
    return [1,2,3,4,5]
print(lista()[1:3])
```

```
variable = lista()
variable[1:4]
```

Return

Una característica interesante, es la posibilidad de devolver valores múltiples separados por comas:

```
def test():
    return "Python", 20, [1,2,3]

test()
('Python', 20, [1, 2, 3])
```

Enviando valores

Enviando valores a una función

Vimos como devolver valores y así comunicar una función con el exterior, ahora enviar información desde el exterior a la función.

Para entender los conceptos más fácilmente vamos a trabajar alrededor de un caso de estudio típico: crear una función que sume dos números y retorne uno en su resultado.

Enviando valores a una función

Lo primero será definir una función la cual denominaremos como **suma** y recibirá 2 números con dos nombres como si fueran dos variables **numero1** y **numero2**, luego retornamos la **suma** entre ambos números.

```
def suma(numero1, numero2):
    return numero1 + numero2
```

Enviando valores a una función

Lo que hacemos para indicar que se reciben valores es **crear dos variables separadas por una coma**. Cuando nosotros llamemos a la función, automáticamente, se le asignarán a estas variables los números que enviemos, siguiendo el mismo orden:

```
>>> r = suma(7, 5)
```

En este caso 7 será la variable numero1 y 5 será la variable numero2

Para pensar

¿Qué ocurriría si lo hiciéramos al revés?

```
>>> r = suma(5, 7)
```

¿Verdadero o falso?

Contesta mediante el chat de Zoom

PARA RECORDAR

Contenido destacado

En este caso 5 será la variable numero1 y 7 será la variable numero2. Hay que tener cuidado por cómo se pasan estos valores a la función, ya que si fuera otra operación matemática podría dar resultados muy distintos, como en una división o potencia.

Momento de una función

Tenemos la definición

```
def suma(numero1, numero2):
    return numero1 + numero2
```

Y la llamada

```
r = suma(7, 5)
```

¿Por qué es importante diferenciar?

Momentos de una función

Durante la definición de la función, las variables o valores se denominan parámetros

```
def suma(numero1, numero2):
    return numero1 + numero2
```

Y durante la llamada se le denominan argumentos, como los argumentos de los scripts.

```
r = suma(7, 5)
```

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Welcome

Escribir una función a la que se le pase una cadena del nombre de una ciudad <ciudad> y muestre por pantalla el saludo ¡hola bienvenidx a <nombre>!.

Duración: **10 minutos**

CODERHOUSE

Media

Escribir una función que reciba una muestra de números en una lista y devuelva su media.

Duración: **10 minutos**

CODERHOUSE

es_multiplo

Crea un programa que le pida dos números enteros al usuario y diga por consola, si alguno de ellos es múltiplo del otro. La función debe llamarse es_multiplo().

Duración: **10 minutos**

CODERHOUSE

#CoderAlert

Encontrarás en la **Guía de Actividades** del curso una actividad para aplicar todo lo aprendido hoy sobre FUNCIONES a tu Proyecto. ¡Será fundamental al momento de realizar tu primera pre entrega en clase N°10!

Actividad Nº 1

Función año bisiesto

Consigna
Realizar una función llamada año_bisiesto:

- ✓ Recibirá un año por parámetro
- ✓ Imprimirá “El año **año** es bisiesto” si el año es bisiesto
- ✓ Imprimirá “El año **año** no es bisiesto” si el año no es bisiesto
- ✓ Si se ingresa algo que no sea número, debe indicar que se ingrese un número.

Información a tener en cuenta:

Se recuerda que los años bisiestos son múltiplos de 4, pero los múltiplos de 100 no lo son, aunque los múltiplos de 400 sí. Estos son algunos ejemplos de posibles respuestas: 2012 es bisiesto, 2010 no es bisiesto, 2000 es bisiesto, 1900 no es bisiesto.

Actividad Nº 1

Función año bisiesto

Formato

- ✓ El documento debe realizarse en Google Docs o mejor aún en Colabs.

Sugerencias

- ✓ En el formulario debe estar el print de pantalla de la consola con el ejercicio resuelto, como así también el código tipeado.

#Codertraining

¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar la **Guía de Ejercicios Complementarios**, donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

¿Preguntas?

Resumen
de la clase hoy

- ✓ Funciones
- ✓ Retorno de valores
- ✓ Envío de valores

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación