PYTHON 360

Carlos Miguel Arias Pérez

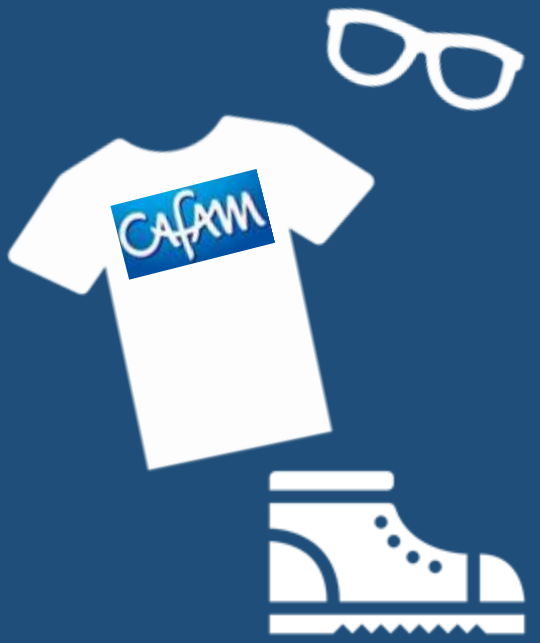
CAFAM

Carlos Miguel Arias Pérez

Ingeniero de Sistemas
Docente universitario

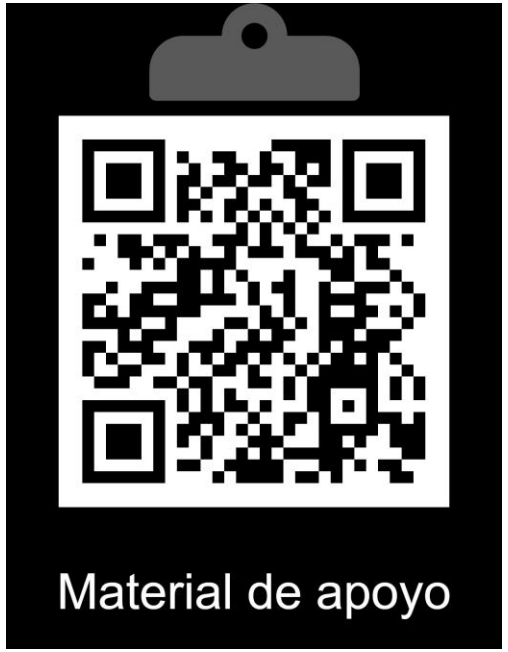
304 5276551

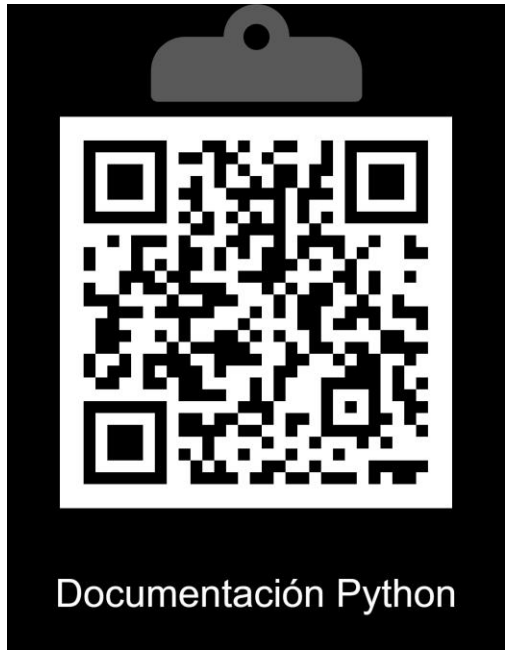
carlos.arias@cet.cafam.edu.co
camarias@cafam.edu.co



Proporcionar a los estudiantes los fundamentos esenciales de Python, incluyendo su sintaxis básica, estructuras de control, manejo de datos y conceptos fundamentales de programación, para que puedan comprender y aplicar principios básicos del desarrollo de software. Este nivel está diseñado para construir una base sólida que permita a los participantes avanzar hacia proyectos más complejos en niveles intermedios y avanzados.

Materiales de apoyo





[Material Python]

https://www.python.org/

Introducción a la Programación yPensamiento Computacional

Programar es dar instrucciones a la computadora.

Pensamiento computacional:

- Dividir problemas en partes.
- Encontrar patrones.
- Crear algoritmos paso a paso.

Ejemplo: Preparar un sándwich (instrucciones simples).

Conceptos fundamentales de programación y algoritmos

Programa = conjunto de instrucciones.

Algoritmo = pasos ordenados para resolver unproblema.

Ejemplo: Encontrar el mayor de dos números.

Pensamiento lógico y resolución de problemas

Un programador:

- - Observa el problema.
- - Divide en pasos.
- - Prueba soluciones.

Ejemplo en Python:

```
numero = 10
if numero % 2 == 0:
    print('Par')
else:
    print('Impar')
```

Ambiente de programación en Python

Necesitamos:

- - Instalar Python.
- - Usar IDLE, IDE, VS Code o Jupyter, Anaconda,
- Colaborate

Ejemplo:

```
print('¡Hola, mundo!')
```

Variables y Tipos de Datos

Variable = espacio para guardar datos.

Ejemplo:
nombre = 'Ana'
Nombre = 'Ana'
edad = 20
esEstudianteActivo = True

Tipos básicos: int, float, str, bool

Manipulación de datos

Ejemplo concatenación:

```
nombre = 'Carlos'
print('Hola ' + nombre)
```

Ejemplo operaciones:

```
x = 5
y = 2
print(x + y) # 7
```

Operadores y Expresiones

Aritméticos: +, -, *, /, %, **

Comparación: ==, !=, <, >, <=, >=

Lógicos: and, or, not

Expresiones simples

Ejemplo:

```
a = 10

b = 5

print(a > b and b > 0) # True
```

Expresiones simples

Operador	Ejemplo	Equivale a
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3

Quiz de Repaso

https://www.w3schools.com/python/exercise.asp?x=xrcise_comments1

https://www.w3schools.com/python/exercise.asp?x=xrcise_variables1

https://www.w3schools.com/python/exercise.asp?x=xrcise_variables_names1

https://www.w3schools.com/python/exercise.asp?x=xrcise_variables_multiple1

https://www.w3schools.com/python/exercise.asp?x=xrcise_variables_output1

https://www.w3schools.com/python/exercise.asp?x=xrcise_operators1

Estructuras de Control de Flujo

Permiten decidir qué instrucciones ejecutar y cuántas veces.

Dos grupos principales:

- Condicionales → tomar decisiones.
- Bucles → repetir acciones.

Condicionales en Python

Sintaxis básica:

```
if condicion:
    # bloque si es verdadero
elif otra_condicion:
    # bloque alternativo
else:
    # bloque si no se cumple nada
```

Ejemplo de Condicional

```
edad = 18
if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
```

La declaración Match de Python

En lugar de escribir muchas declaraciones if..else, se puede utilizar la declaración match.
match selecciona uno de muchos bloques de código que se ejecutarán.

Sintaxis

```
match expression:
    case x:
      code block
    case y:
      code block
    case z:
      code block
```

La declaración Match de Python

Ejemplo

```
day = 4
  match day:
    case 1:
      print("Monday")
    case 2:
      print("Tuesday")
    case 3:
      print("Wednesday")
    case 4:
      print("Thursday")
    case 5:
      print("Friday")
    case 6:
      print("Saturday")
    case 7:
      print("Sunday")
```

La declaración Match de Python

Valor predeterminado

```
day = 4
  match day:
    case 6:
      print("Today is Saturday")
    case 7:
      print("Today is Sunday")
    case _:
      print("Looking forward to the
  Weekend")
```

La declaración Match de Python

Combinar valores

Utilice el carácter de barra vertical | como operador o en la evaluación de caso para verificar si hay más de una coincidencia de valores en un caso.

Ejemplo

```
month = 5
day = 4
match day:
  case 1 | 2 | 3 | 4 | 5 if month == 4:
    print("A weekday in April")
  case 1 | 2 | 3 | 4 | 5 if month == 5:
    print("A weekday in May")
  case _:
    print("No match")
```

El bucle while

Con el bucle while podemos ejecutar un conjunto de sentencias siempre que una condición sea verdadera.

```
i = 1

while i < 6:
    print(i)
    i += 1
```

El bucle while – uso de Break

Con la sentencia break podemos detener el bucle incluso si la condición while es verdadera

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

El bucle while – uso de continue

Con la sentencia **continue** podemos detener la iteración actual y continuar con la siguiente

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

El bucle while – uso de else

Con la declaración **else** podemos ejecutar un bloque de código una vez cuando la condición ya no sea verdadera

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less
than 6")
```

El bucle For

Un bucle for se utiliza para iterar sobre una secuencia (que puede ser una lista, una tupla, un diccionario, un conjunto o una cadena). Con el bucle for podemos ejecutar un conjunto de sentencias, una vez para cada elemento de una lista, tupla, conjunto, etc.

```
fruits =
  ["apple", "banana", "cherry"]
  for x in fruits:
    print(x)

  for x in "banana":
    print(x)
```

El bucle For – uso de Break

```
fruits = ["apple", "banana", "cherry"]
 for x in fruits:
   print(x)
   if x == "banana":
     break
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

El bucle For – uso de Continue

```
fruits =
  ["apple", "banana", "cherry"]
  for x in fruits:
    if x == "banana":
      continue
    print(x)
```

El bucle For - uso de Range

Para recorrer un conjunto de código una cantidad específica de veces, podemos usar la función range().
La función range() devuelve una secuencia de números, comenzando desde 0 de manera predeterminada, y se incrementa en 1 y finaliza en un número especificado.

```
for x in range(6):
    print(x)
```

```
for x in range(2, 6):
  print(x)
```

```
for x in range(2, 30, 3):
  print(x)
```

El bucle For - uso de else

Especifica un bloque de código que se ejecutará cuando finalice el bucle.

```
for x in range(6):
    print(x)
  else:
    print("Finally finished!")
```

El bucle For - uso de pass

Los bucles no pueden estar vacíos, pero si por alguna razón se tiene un bucle sin contenido, se usa **pass** para evitar obtener un error.

```
for x in [0, 1, 2]:
    pass
```

Quiz de repaso

https://www.w3schools.com/python/exercise.asp?x=xrcise_conditions1

https://www.w3schools.com/python/exercise.asp?x=xrcise_match1

https://www.w3schools.com/python/exercise.asp?x=xrcise_while_loops1

https://www.w3schools.com/python/exercise.asp?x=xrcise_for_loops1

Aplicaciones de Control de Flujo

- Clasificar números como pares o impares.
- Calcular sumas, promedios y factoriales.
- Recorrer listas de datos.
- Simular menús y procesos repetitivos.

Funciones

Una función es un bloque de código que solo se ejecuta cuando se lo llama. Puede pasar datos, conocidos como parámetros, a una función. Una función puede devolver datos como resultado.

Creando una función

En Python, una función se define utilizando la def palabra clave:

```
def my_function():
    print("Hello from a function")
```

Llamar a una función

Para llamar a una función, utilice el nombre de la función seguido de paréntesis:

```
def my_function():
    print("Hello from a function")

my_function()
```

Argumentos

La información se puede pasar a las funciones como argumentos. Los argumentos se especifican después del nombre de la función, entre paréntesis. Puedes agregar tantos argumentos como quieras, simplemente separándolos con una coma.

```
def my_function(fname):
    print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Número de Argumentos

Por defecto, una función debe llamarse con el número correcto de argumentos. Esto significa que, si la función espera dos argumentos, debe llamarse con dos argumentos, ni más ni menos.

```
def my_function(name, apellido):
    print(name + " " + apellido)

my_function("Juan", "Pérez")
```

Argumentos arbitrarios, *args

Si no sabe cuántos argumentos se pasarán a su función, agregue un *antes del nombre del parámetro en la definición de la función. De esta manera, la función recibirá una tupla de argumentos y podrá acceder a los elementos en consecuencia:

```
def my_function(*kids):
    print("The youngest child is
  " + kids[2])

  my_function("Emil", "Tobias",
  "Linus")
```

Argumentos de clave = valor

ambién puede enviar argumentos con la sintaxis clave = valor .
De esta manera el orden de los argumentos no importa.

```
def my_function(child3, child2,
  child1):
    print("The youngest child is " +
  child3)

  my_function(child1 = "Emil", child2
  = "Tobias", child3 = "Linus")
```

Argumentos de clave = valor arbitrarios

Si no sabe cuántos argumentos de palabras clave se pasarán a su función, agregue dos asteriscos: **antes del nombre del parámetro en la definición de la función.
De esta manera, la función recibirá un diccionario de argumentos y podrá acceder a los elementos en consecuencia:

```
def my_function(**kid):
    print("Su apellido es " +
  kid["apellido"])

  my_function(nombre = "Tobias",
  apellido = "López")
```

Valor del parámetro predeterminado

El siguiente ejemplo muestra cómo utilizar un valor de parámetro predeterminado.
Si llamamos a la función sin argumentos, utiliza el valor predeterminado:

```
def my_function(country = "Norway"):
    print("Yo nací en " + country)

  my_function("Sweden")
  my_function("India")
  my_function()
  my_function("Brazil")
```

Valores de retorno

Para permitir que una función devuelva un valor, utilice la declaración return :

```
def my_function(x):
    return 5 * x

  print(my_function(3))
  valor = my_function(5)
  print(my_function(9))
```

Quiz de repaso

https://www.w3schools.com/python/exercise.asp?x=xrcise_functions1

Cierre y recomendaciones

Programar es pensar paso a paso.

Python es sencillo y poderoso.

La clave: practicar todos los días.

¡Gracias!