El futuro digital es de todos

MinTIC

FUNCIONES

Hechos QUE CONECTAN

Universidad Industrial de Santander

UIS

MisionTIC 2022

4.1. INTRODUCCIÓN

Una función "es un bloque de código que tiene asociado un nombre y que recibe o no una serie de argumentos como entrada, para luego seguir una serie de instrucciones que tienen como finalidad llevar a cabo una tarea específica y retornar un valor". John Snowden (2020) (Python For Beginners: A Practical Guide For The People Who Want to Learn Python The Right and Simple Way. Independently published)

Algunos ejemplos de funciones en la vida diaria son:

- Sumar dos números.
- Preparar una comida.
- Ir de un punto a otro en la ciudad.
- Operar una máquina.
- Contar las palabras dentro de un texto.

Las funciones permiten dividir un código u aplicación en pequeñas tareas que hace parte de la solución a un problema más grande. Un ejemplo de división de tareas en pequeñas tareas son:

- Las tareas que se llevan a cabo en torno a un proyecto de una empresa. Se requiere para llevar a cabo un proyecto de diferente tipo de personales. Por mencionar algunos, los que llevan el presupuesto del proyecto, los científicos que determinan que las condiciones para realizar el proyecto sean las óptimas, y los ingenieros que llevan a cabo el desarrollo de lo que va a resultar del proyecto.

4.2. COMPOSICIÓN DE UNA FUNCIÓN

Las funciones poseen una estructura básica y se caracterizan por estar compuestas de una serie de elementos. Estos elementos son:

Nombre:

Algunos ejemplos de nombres de funciones:

- Si se desea realizar la suma de dos números, entonces lo más lógico es que esta función se llame **suma**.
- Si se desea buscar entre un conjunto de números cuáles son números primos, entonces podemos llamar esta función **numeros_primos** o **prime_number**. De esta manera, quien lea el código sabrá cuál es la tarea de dicha función sin necesidad de buscar en el contenido.

Argumentos:

Algunos ejemplos de argumentos de funciones:

- Si queremos construir una función que me retorne cualquier número entero, no es necesario que esta función reciba argumentos de entrada.

- Si lo que se desea construir es una función que retorne si un número es menor que cero, entonces se debe recibir como parámetro de entrada cuál es ese número.

Secuencia de instrucciones:

Algunos ejemplos de secuencia de instrucciones de funciones:

- Si se desea construir una función que determine cuál es la suma de dos números, estos dos números deben ser recibidos como parámetros de entrada ,y dentro de la función, se debe llevar a cabo la respectiva suma de ellos. Este proceso de suma es lo que corresponde a la **secuencia de instrucciones**.

- Si se desea construir una función que me calcule la temperatura en grados Fahrenheit a partir de la temperatura en grados celsius (argumentos de entrada), su **secuencia de instrucciones** estará conformada por el proceso que se use para hacer la conversión de grados celsius a grados Fahrenheit.

El retorno:

Algunos ejemplos de retornos de funciones:

- Si volvemos al caso en que la función realice la suma de dos números, esta retornará el valor que resulta de la suma de esos dos números.
- Si volvemos al caso en que la función retorna cualquier número entero, entonces el retorno de esa función será el valor de ese número.

A continuación, son mostradas algunas estructuras ejemplo de funciones.

Un ejemplo básico de los elementos de una función se puede ver a continuación:

```
define <nombre de función>(<argumento 1>) {
    <secuencia de instrucciones>
    retorna <resultado>
}
```

Los argumentos son necesarios dentro de la función. Si estos argumentos no son necesarios, entonces es mejor definir la función sin argumentos. Por ello, una función puede no recibir argumentos:

```
define <nombre de función>() {
    <secuencia de instrucciones>
    retorna <resultado>
}
```

Si la tarea que va a realizar la función no resulta en un valor que se requiera fuera de la función, entonces no es necesario que esta función retorne un valor. Por lo que una función puede no retornar un valor:

```
define <nombre de función>(<argumento 1>) {
    <secuencia de instrucciones>
}
```

Si la función que se está desarrollando requiere más de un argumento para llevar a cabo la tarea para la cual se quiere construir, entonces es necesario pasarle todos los argumentos que sean requeridos. Entonces, las funciones pueden recibir más de un argumento:

```
define <nombre de función>(<argumento 1>, <argumento 2>, <argumento 3>) {
    <secuencia de instrucciones>
    retorna <resultado>
}
```

El nombre que se le da a la función es `<nombre de función>`, sus argumentos de entrada son `<argumento 1>`, `<argumento 2>`, …, las instrucciones que va a ejecutar son `<secuencia de instrucciones>`, y el retorno es `<resultado>`.

Dentro de este concepto de funciones se maneja también un término asociado con la vida útil de una función, el cual es el ciclo de vida de una función. Este ciclo de vida se resume en que la función es definida para luego ser usada, y finalmente, morir.

4.3. FUNCIÓN Y MÓDULOS PREDEFINIDOS

4.3.1. Funciones predeterminadas

Algunas funciones predeterminadas en el lenguaje Python son:

4.3.1.1 Función type()

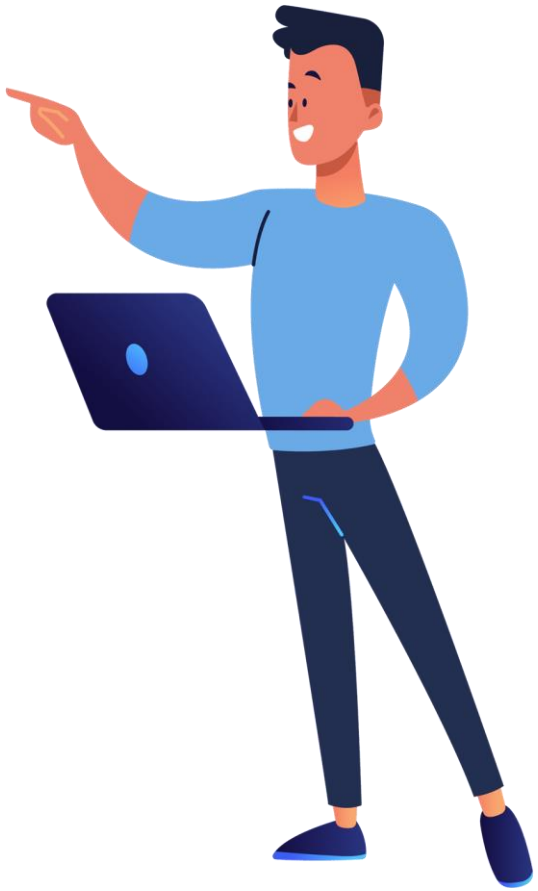
La función type es una función que retorna el tipo de clase de un objeto que es pasado como argumento.

Sintaxis:

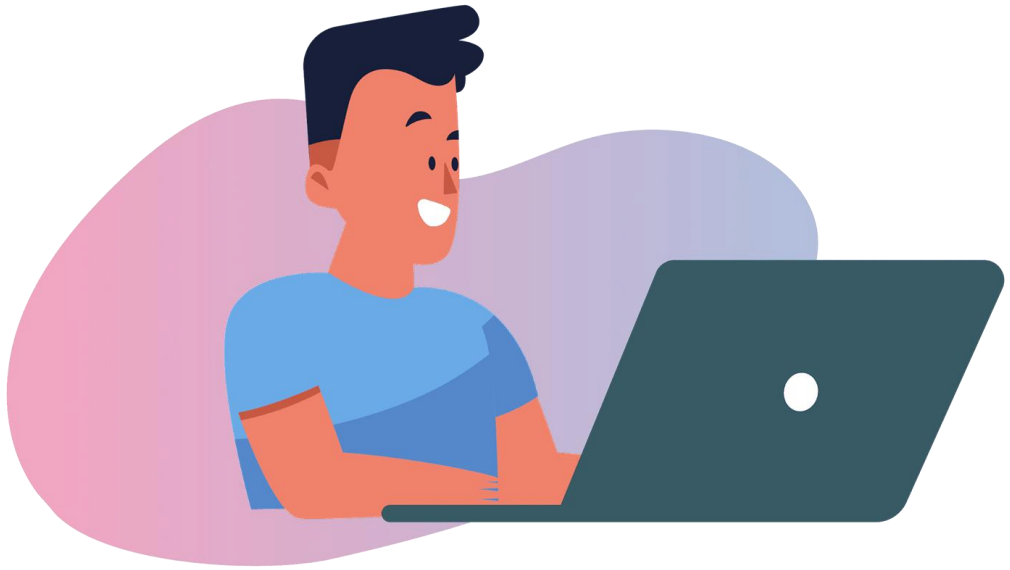
```
type(objeto)
```

Parámetros:

objeto: objeto al cual se le desee conocer el tipo de clase.



4.3.1.2 Función max()



La función max es una función que retorna el máximo valor de una serie de parámetros de igual tipo o de una lista.

Sintaxis:

```
max(a,b,c,d,…)
```

Parámetros:

a,b,c,…: datos del mismo tipo.

4.3.1.3 Función sorted()

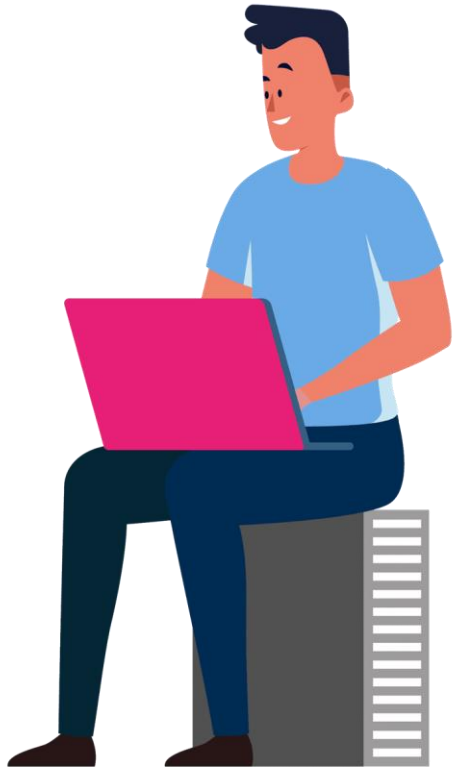
La función sorted es una función que retorna una lista ordenada de datos a partir de unos datos en una variable iterable.

Sintaxis:

```
sorted(objeto)
```

Parámetros:

objeto: datos de entrada de tipo variable iterable.



4.3.1.4 Función print()

La función print es una función que imprime el objeto dado en la salida estándar del dispositivo.

Sintaxis:

```
print(objeto)
```

Parámetros:

objeto: objeto para imprimir.

4.3.1.5 Función len()

La función len es una función que retorna el número de elementos en un objeto.

Sintaxis:

```
len(objeto)
```

Parámetros:

objeto: secuencia de datos (cadena de caracteres, tuplas, diccionarios o conjuntos).

4.3.1.6 Función input()

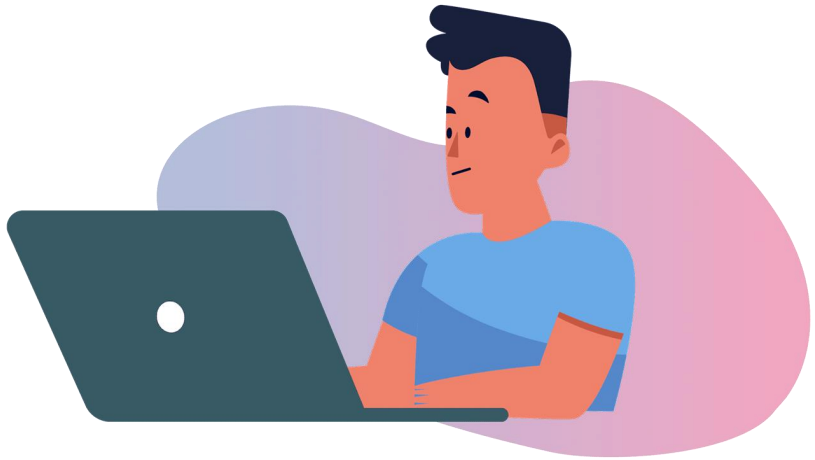
La función input es una función que permite obtener texto introducido usando el teclado.

Sintaxis:

```
variable = input(prompt)
```

Parámetros:

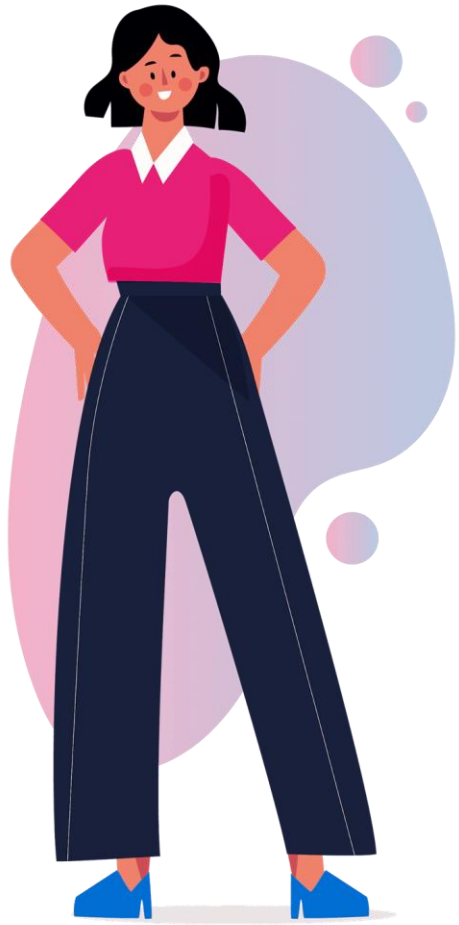
prompt: es el mensaje que será mostrado antes de que se ingrese alguna información por medio del teclado. Esa información es almacenada dentro de la variable luego de recibir una orden emitida al presionar la tecla Enter.



4.3.2 Módulos predefinidos

Para ser usado en un código, un módulo requiere de su importación, lo cual es logrado mediante las palabras reservadas `from` e `import`.

La biblioteca estándar de Python viene con un conjunto de módulos, cada uno de ellos desarrollados para llevar a cabo diferentes tipos de tareas. A continuación, se muestra una lista de alguno de estos módulos y sus funciones.



4.3.2.1 Módulo os

Este módulo provee funciones que permiten al desarrollador interactuar con el sistema operativo; como por ejemplo, este módulo tiene la función getcwd() que permite conocer la ruta del directorio en el cual te encuentras.

```
>>> import os
>>> os.getcwd()
>>> /ruta/del/directorio/donde/te/encuentras/
```

Este módulo posee también un submódulo, `path`, que permite acceder a funcionalidades como conocer si una ruta es un archivo, `os.path.isfile()`, o si la ruta es un directorio, `os.path.isdir()`.

Conocer más del módulo os

Conocer más del submódulo path

4.3.2.2 Módulo glob

Este módulo permite encontrar todos los nombres de rutas que coincidan con cierto patrón específico. Para encontrar las rutas de todos los archivos de Python, archivos con extensión `.py`, que se encuentran en el directorio actual, se hace de la siguiente manera:

```
>>> from glob import glob
>>> glob.glob('.' + '/*.py')
```

que significa que se buscarán las rutas de los archivos `.py` que se encuentran en la ruta `'.'`.

4.3.2.3 Módulo sys

Este módulo permite tener acceso a variables y funciones relacionadas con el intérprete de Python; como por ejemplo, las variables relacionadas con los parámetros que se le pasan a un código en Python por línea de comando. Los códigos en Python son ejecutados por medio de la siguiente línea de comando:

```
>> python archivo.py
```

donde adicionalmente se le pueden pasar argumentos de la siguiente manera:

```
>>> python archivo.py arg1 arg2 arg3
```

Estos argumentos, `archivo.py`, `arg1`, `arg2` y `arg3`, son almacenados en una variable, la cual se puede acceder desde dentro del código como:

```
import sys

argumentos = sys.argv

print(argumentos)
```

Este fragmento de código imprime cada una de las variables pasadas al momento de ejecución del script de Python.

Conocer más del módulo sys

4.3.2.4 Módulo math

Este módulo permite el acceso a funciones matemáticas dentro de la biblioteca de C. Por ejemplo, se requiere el uso de las funciones trigonométricas. Estas se pueden acceder de la siguiente manera:

```
>>> from math import sin, cos, tan, pi
>>> sin(pi/3)
>>> 0.8660254037844386
>>> cos(pi/3)
>>> 0.5000000000000001
>>> tan(pi/4)
>>> 0.9999999999999999
```

Se pueden emplear las funciones `sin`, `cos`, `tan`, entre otras y además, se puede emplear parámetros como el número `pi`. Existen muchos otros tipos de funciones dentro de este módulo que puedes explorar; como por ejemplo, la función raíz cuadrada, `sqrt`, exponenciación, `exp`, valor absoluto, `fabs`, entre otras.

Conocer más del módulo math

4.3.2.5 Módulo random

Este módulo provee herramientas para la generación de elementos aleatorios. Se importa de la siguiente manera:

```
>>> import random
```

Para generar un número aleatorio se hace de las siguientes maneras:

- Si se desea obtener un valor aleatorio tipo entero entre dos valores a y b, se hace usando la orden `random.randint(a, b)`.
- Si se desea obtener un valor flotante aleatorio entre dos valores a y b se puede hacer usando la orden `random.uniform(a, b)`.
- Si se desea elegir de manera aleatoria un ítem de una lista de ítems, se puede hacer usando la orden `random.choice(lista)`.

Existen otras funciones que se pueden explorar y usar, las cuales se encuentran dentro de este módulo.

Conocer más del módulo random

4.3.2.6 Módulo datetime

Este módulo proporciona los elementos necesarios para la manipulación de fechas y horas. Si se requiere conocer cuál es la fecha actual, se puede hacer de la siguiente manera:

Suponiendo que hoy es 11 de febrero de 2022, para obtener la fecha actual, se hace de la siguiente manera:

```
>>> import datetime
>>> fecha = datetime.date.today()
```

En fecha es almacenada la fecha actual, esta se puede visualizar de la siguiente manera:

```
>>> fecha
>>> datetime.date(2022, 2, 11)
```

Se puede acceder a el día, el mes y el año, usando los atributos `day`, `month` y `year`, de la siguiente manera:

```
>>> fecha.day
>>> 11
>>> fecha.month
>>> 2
>>> fecha.year
>>> 2022
```

Si se desea conocer cuál es la fecha y hora actual, el módulo `datetime` tiene un submódulo con el mismo nombre, dentro del cual podemos encontrar el método `now()`, que retorna el valor de la fecha y hora actual. Esto se puede hace de la siguiente manera:

```
>>> from datetime import datetime
>>> fecha_hora = datetime.now()
>>> fecha_hora
>>> datetime.datetime(2022, 2, 11, 19, 19, 35, 472337)
```

Se puede acceder también al día, el mes, el año, la hora, los minutos y los segundos, usando los atributos `day`, `month`, `year`, `hour`, `minute`, y `second`, de la siguiente manera:

```
>>> fecha_hora.day
>>> 11
>>> fecha_hora.month
>>> 2
>>> fecha_hora.year
>>> 2022
>>> fecha_hora.hour
>>> 19
>>> fecha_hora.minute
>>> 23
>>> fecha_hora.second
>>> 35
```

Se pueden crear datos de tipo `datetime` mediante el método `date(year, month, day)` , que se encuentra dentro del módulo `datetime`, esto se hace de la siguiente manera:

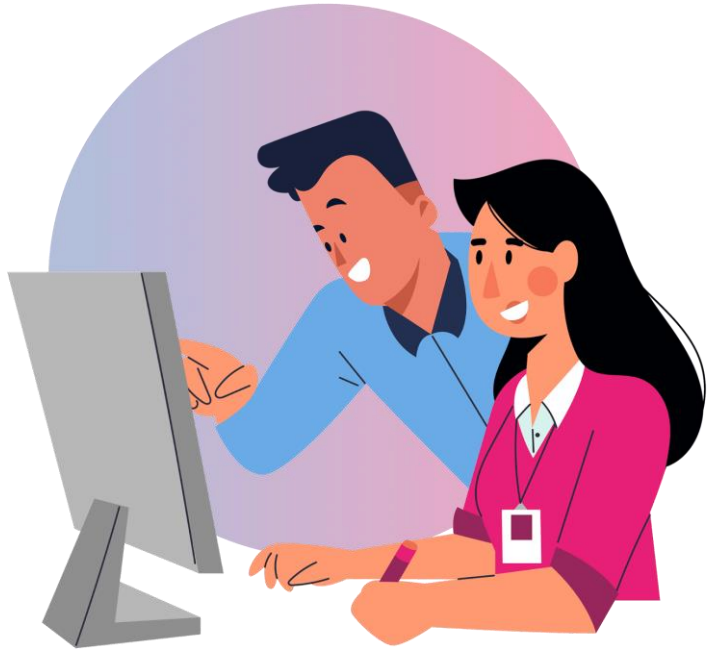
```
>>> import datetime
>>> fecha = datetime.date(2022, 1, 1)
>>> fecha
>>> datetime.date(2022, 1, 1)
```

Adicional a estas funciones, existen otras funciones que se pueden explorar dentro de este módulo.

Conocer más del módulo datetime

4.4. FUNCIONES O MÉTODOS PARA CARACTERES Y CADENAS

A continuación, son mostrados algunos métodos para el manejo de cadena de caracteres:

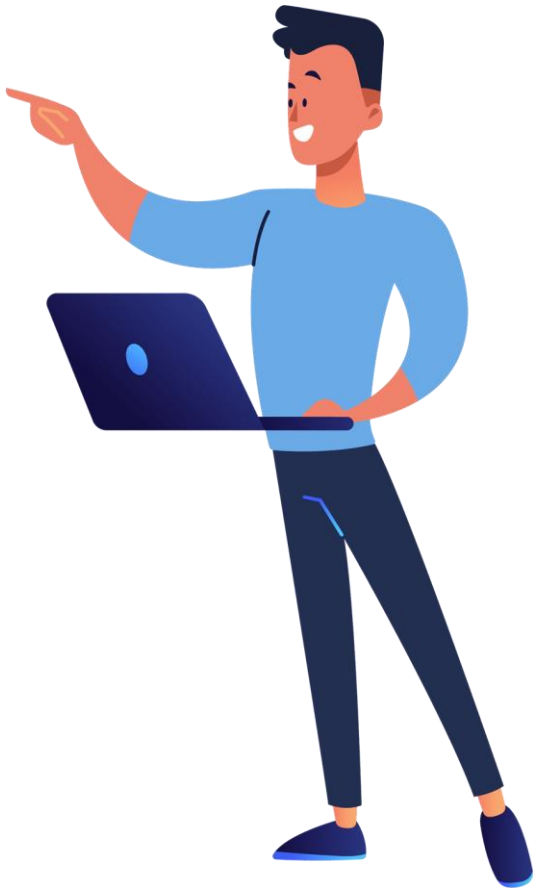


4.4.1 Método capitalize()

Este método muestra la primera letra de la cadena de caracteres en mayúscula.

Ejemplo:

```
>>> caracteres = 'hola'
>>> caracteres.capitalize()
>>> 'Hola'
```



4.4.2 Método count()

Este método indica cuantas veces se repite un valor dentro de una cadena de caracteres.

Ejemplo:

```
>>> caracteres = 'asombroso'
>>> valor = 'o'
>>> caracteres.count(valor)
>>> 3
```

Este método recibe adicionalmente un parámetro de entrada, el cual es el valor que se pretende buscar dentro de la línea de caracteres. Como se puede apreciar, la cadena de caracteres `'asombroso'` tiene 3 caracteres `'o'`, el cual es el resultado mostrado.

4.4.3 Método find()

Este método permite buscar un valor específico dentro de una cadena de caracteres y retorna la posición donde fue encontrado.

Ejemplo:

```
>>> caracteres = "Hola a todos, bienvenidos a este curso"
>>> valor = "este"
>>> caracteres.find(valor)
>>> 28
```

El valor retornado es 28,ya que si contamos desde el primer carácter de la línea de caracteres definida, vamos a encontrar que la palabra `"este"` se encuentra a partir de la letra 28, teniendo en cuenta que se contabiliza desde cero.

4.4.4 Método split()

Este método permite dividir una cadena de caracteres dependiendo de un separador, el cual puede ser cualquier carácter o conjunto de caracteres. Si no se le especifica cuál es el separador, este es tomado como el espacio en blanco entre conjunto de caracteres o palabras dentro de la cadena de caracteres. Este método retorna una lista de trozos de la cadena de caracteres tomados de acuerdo al separador.

Ejemplo:

```
>>> caracteres = "1 2 3 4 5 6"
>>> caracteres.split()
>>> [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’]
```

Ejemplo:

```
>>> caracteres = "datos.txt"
>>> caracteres.split(".")
>>> ["datos", "txt"]
```

4.4.5 Método format()

Este método permite darle formato a valores específicos e insertarlos en un marcador de posición dentro de una cadena de caracteres. El marcador de posición es definido usando corchetes, `{}`. Este método retorna la cadena de caracteres formateada. En algunas ocasiones, este método es usado para generar salidas.

Ejemplo:

La manera más simple de utilizar este método es la siguiente:

```
>>> cadena = "Hola {}, ¡bienvenido a este curso de {}!"
>>> nombre = "Juan"
>>> curso = "Python"
>>> cadena.format(nombre, curso)
>>> Hola Juan, ¡bienvenido a este curso de Python!
```

Ejemplo:

Dentro de la cadena de caracteres, también podemos identificar la posición de cada argumento que se pasa a formar, esto por medio de números, de la siguiente manera:

```
>>> cadena = "Hola {1}, ¡bienvenido a este curso de {0}!"
>>> nombre = "Juan"
>>> curso = "Python"
>>> cadena.format(curso, nombre)
>>> Hola Juan, ¡bienvenido a este curso de Python!
```

En este caso, el primer argumento, curso, debe ir ubicado en `{0}` y el segundo argumento, nombre, debe ir ubicado en `{1}`.

4.4.6 Método join()

Este método permite concatenar un objeto iterable, cuyos elementos son caracteres o cadena de caracteres, con un delimitador definido por el usuario. Este método toma cada uno de los elementos del objeto iterable y los concatena en una cadena de caracteres separando cada elemento con el delimitador definido.

Ejemplo:

Un ejemplo con una cadena de caracteres podría ser:

```
>>> letras = "abcdefg"
>>> delimitador = ","
>>> delimitador.join(letras)
>>> "a,b,c,d,e,f,g"
```

Las cadenas de caracteres son objetos iterables, y el método `join()` toma cada uno de sus elementos y los concatena en una nueva cadena de caracteres separándolos por el delimitador que en este caso es una coma.

Ejemplo:

Un ejemplo usando una lista es:

```
>>> letras = ["Hola", "como", "estas"]
>>> delimitador = " "
>>> delimitador.join(letras)
>>> "Hola como estas“
```

En este ejemplo, el delimitador es un espacio en blanco. Se toma cada elemento de la lista y es concatenado en una cadena de caracteres, separando cada uno de los elementos con un espacio en blanco. De esta misma manera, se puede llevar a cabo la concatenación usando conjuntos o tuplas.

4.4.7 Método replace()

Este método permite crear una nueva cadena de caracteres mediante el reemplazo de alguna parte de otra cadena de caracteres.

Ejemplos:

Un ejemplo del uso de este método es el siguiente:

```
>>> frase = "Los patos vuelan"
>>> frase.replace("vuelan", "nadan")
>>> "Los patos nadan“
```

En este caso se genera una nueva cadena de caracteres a partir del reemplazo o modificación de cierta parte de otra cadena de caracteres. En el ejemplo tenemos la frase `"Los patos vuelan"`, y se quiere crear otra cadena de caracteres que indique que los patos nadan, por lo que en la frase original se cambia la palabra `"vuelan"` por la palabra `"nadan"` . El primer parámetro de entrada del método es lo que se quiere modificar y el segundo parámetro es por el cual se quiere modificar.

4.4.8 Método casefold()

Este método retorna una nueva cadena de caracteres eliminando todas las distinciones de casos que presente, de tal manera que es usado para hacer comparaciones de cadena de caracteres.

Ejemplo:

Un ejemplo para el uso de este método podría ser:

```
>>> palabra1 = "Estado"
>>> palabra2 = "estado"
>>> palabra1.casefold() == palabra2.casefold()
>>> True
```

En este ejemplo se comparan dos cadenas de caracteres. Estas cadenas de caracteres son palabras iguales, pero una de ellas contiene una letra en mayúscula y la otra no. El método `casefold()` permite comparar estas dos palabras sin tener en cuenta el hecho de que una de ellas tiene una letra en mayúscula.

Como se puede observar en los ejemplos anteriores, los métodos para los datos tipo cadena de caracteres, son llamados haciendo uso de la cadena de caracteres definida junto con el operador punto. Cada uno de estos métodos opera sobre los datos de la variable sin modificarla. Existen muchos otros métodos con diversas funcionalidades que permiten también el manejo de cadenas de caracteres.

4.5. FUNCIONES PSEUDOCÓDIGO

A continuación se muestra un ejemplo del pseudocódigo para una función que permite contar, de cada elemento dentro de una cadena de caracteres, cuantas veces aparece dicho elemento dentro de la cadena de caracteres, y retorna una lista con tales valores:

```
function count_each_character(input_string)
    string_length ← length of input_string
    counter_array ← empty array of length string_length
    for i ← 0 to i ← string_length - 1 do
        character ← input_string(i)
        counter ← 0
        for j ← 0 to j ← string_length - 1 do
            if input_string(i) == input_string(j) then
                counter ← counter + 1
            end if
        end for
        counter_array(i) ← counter
    end for
    return counter_array
end function
```

El pseudocódigo anterior se desglosa de la siguiente manera:

- **function** `count_each_character(input_string)`

La función lleva como nombre `count_each_character` y toma como dato de entrada una cadena de caracteres almacenada en `input_string`.

- `string_length ← length of input_string`

  `counter_array ← empty array of length string_length`

Se define dentro de la función dos variables, una de tipo entero, `string_length`, para almacenar el tamaño de la cadena de caracteres, y la otra es un arreglo, `counter_array`, del mismo tamaño que `input_string`.

- **for** i ← 0 to i ← string_length – 1 **do**

    . . .

    for j ← 0 to j ← string_length – 1 **do**

Se procede creando un doble ciclo `for`, ambos iterando desde cero hasta el valor `string_length-1`.

- character ← input_string(i)

counter ← 0

El primer ciclo `for` permite definir cada uno de los caracteres que componen el arreglo de caracteres mediante la variable `character`, y dentro de este se define un contador (`counter`) que se hace cero justo antes de empezar el segundo ciclo `for`.

- **if** input_string(i) == input_string(j) **then**

El segundo ciclo `for` permite nuevamente iterar sobre cada uno de los caracteres que componen la cadena de caracteres, los cuales son comparados con la variable `character` por medio de un condicional `if`.

- `counter ← counter + 1`

Si el condicional se cumple entonces el contador agrega una unidad a su valor, de tal manera que cuando se finaliza el segundo ciclo `for`, el valor de `counter` es igual al número de veces que el carácter almacenado en `character` aparece dentro de la cadena de caracteres.

- `counter_array(i) ← counter`

El valor de `counter` es almacenado en el arreglo `counter_array` de acuerdo a cada posición de los caracteres en la cadena de caracteres.

- **`return`** `counter_array`

Al finalizar el segundo ciclo `for` la función habrá hecho esta misma tarea sobre cada carácter de la cadena de caracteres para finalmente retornar el valor almacenado en `counter_array`.

Existen algunas cosas que no se han tenido en cuenta en este pseudocódigo de la función `count_each_character`, como es el caso de no contar los caracteres repetidos dentro de la cadena de caracteres. Esto se ha hecho de manera intencional, ya que se busca que la lógica sea más sencilla, de tal manera que se entienda la idea de cada elemento dentro de la función.

4.6. FUNCIONES Y ARGUMENTOS

4.6.1 Definición de una función en Python

A continuación, se muestran algunos ejemplos de definición de funciones en Python.

Ejemplo 1:

Pensemos en una función hipotética para convertir unidades de velocidad, `kms_to_ms()`, que permite transformar unidades en **km/h** a unidades en **m/seg**. La conversión es simple, se requiere tener en cuenta que **1 km** son **1000 m** y que **1 h** equivale a **3600 seg**. A partir de esto, mediante regla de tres, tenemos que la conversión es la siguiente:

```
velocidad en m/seg ← (velocidad en km/h) * (1000m / 1km) * (1h / 3600 seg)
```

Entonces definimos la función `kms_to_ms()` como:

```
def kms_to_ms(velocity_kms):
    kilometer_in_meter = 1000
    seconds_in_hour = 3600
    velocity_ms = velocity_kms * kilometer_in_meter / seconds_in_hour
    return velocity_ms
```

donde el valor que se retorna dentro de la variable `velocity_ms` es el valor de la velocidad en metros por segundos.

Ejemplo 2:

Otro ejemplo de funciones puede ser una función que emita un mensaje de acuerdo a un nombre de un usuario y retorna la fecha y hora en la que ese mensaje fue emitido. Esta función recibe **2** parámetros de entrada, uno es una cadena de caracteres donde es almacenado un nombre, `name`, y el otro es otra cadena de caracteres donde se almacena un mensaje, `msg`. Dentro de la función se imprime el nombre seguido de dos puntos seguido del mensaje que se quiere emitir. Esto lo podemos asemejar a un algoritmo para visualizar el mensaje de un usuario de una aplicación de chat. Esta función se puede definir de la siguiente manera:

```
from datetime import datetime


def chat(name, msg):
    date = datetime.now()
    print("{}: {}".format(name, msg))
    return date
```

La función `chat()` hace uso del módulo `datetime` para acceder a la información de la fecha y hora actual, esto mediante el método `now()`. La información de la fecha y hora es almacenada es la variable date. Luego se imprime el nombre del usuario junto con el mensaje emitido. Finalmente, se retorna la información de la fecha y hora.

El llamado de la función `chat()` se hace de la siguiente manera:

```
>>> chat('Pedro', 'Hola,  como estas?')
>>> Pedro: Hola, ¿cómo estás?
```

Si se ejecuta esta función solamente con el nombre del usuario, esta retorna error. Se puede hacer que el segundo argumento tenga un valor predeterminado, haciéndose de la siguiente manera:

se cambia en la función, esto

```
def chat(name, msg):
```

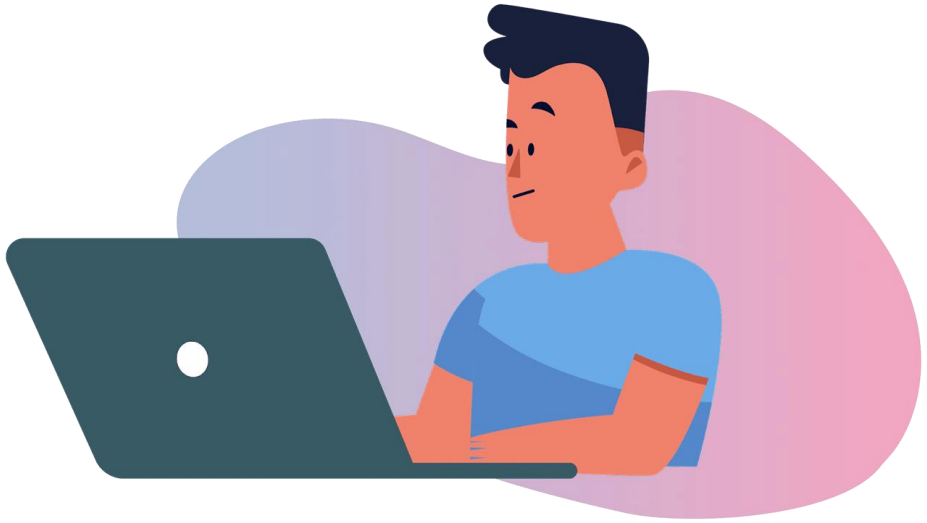
por esto

```
def chat(name, msg = 'Hola!'):
```

quedando la función definida como

```
from datetime import datetime

def chat(name, msg = 'Hola!'):
    date = datetime.now()
    print("{}: {}".format(name, msg))
    return date
```



En este caso `msg = 'Hola!'` significa que este será el mensaje predeterminado, de tal manera que si ejecutamos la función sin el segundo argumento, esta no arrojará un error, sino que visualizará el mensaje predeterminado, como se muestra a continuación:

```
>>> chat('Pedro')

>>> Pedro: Hola!
```

Otra manera de pasar argumentos es mediante el uso de las palabras clave usadas para nombrar los argumentos de la función; por ejemplo, en el ejemplo 2 se puede hacer uso de los nombres `name` y `msg` para pasar los argumentos de la función. De esta manera, lo que se hace es evitar colocar los argumentos en el orden en que aparecen al inicio de la definición de la función. Un ejemplo de esto es:

```
>>> chat(msg = 'Hola, como estas?', name = 'Pedro')

>>> Pedro: Hola, ¿cómo estás?
```

Cabe mencionar que, una función puede no retornar un valor, como también puede retornar uno o múltiples valores. En el caso en que se retornen múltiples valores, lo que se hace es que después de la palabra `return`, se colocan los datos que se retornan separados por coma. Esto es:

```
return a, b, c
```

donde `a, b, c` son los valores a retornar.

4.6.2 Funciones recursivas

Un ejemplo de función recursiva es la función factorial. El factorial de un número entero y positivo, es el número multiplicado por los números que le preceden. Esto es el factorial de 1 es 1 el factorial de 2 es 2x1, el factorial de 3 es 3x2x1, y así sucesivamente. Una función recursiva que permite resolver este problema es la siguiente:

```
def factorial(n):
    if n == 1:
        return n
    elif n < 1:
        return ("No existe el factorial de números negativos!")
    else:
        return n*factorial(n-1)
```

Esta función toma como parámetro de entrada un número, al cual se le evaluará el factorial de la siguiente manera: Si el número `n` es cualquier número entero positivo, se buscará de manera recursiva el factorial de este número. La función accede a sí misma cuantas veces sea necesario, hasta que se cumpla la condición interna de la función. Si el número `n` es menor que 1, el factorial no existe y la función retorna un aviso.

Material de estudio complementario

Analogía de pseudocódigos a lenguaje Python

Funciones predefinidas en Python

Funciones en Python

Argumentos de una función

Módulos en Python

Referencias bibliográficas

[1] Hunt, J. (2021). A Beginners Guide to Python 3 Programming.

[2] Gowrishankar, S., & Veena, A. (2018). Introduction to Python Programming. CRC Press.

[3] John Snowden (2020). Python For Beginners: A Practical Guide For The People Who Want to Learn Python The Right and Simple Way. Independently published.

El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

CICLO 1
EJE TEMÁTICO 4
FUNCIONES

Universidad Industrial de Santander

MisionTIC 2022