IN1444C
Taller de Python para Ciencia de Datos

Pedro Gómez

Ingeniería Civil Informática
Departamento de Ingeniería Informática
Facultad de Ingeniería
Universidad Católica de la Santísima Concepción
pgomez@ucsc.cl

Introducción

- Python es uno de los lenguajes de programación más populares y es el más ampliamente utilizado en el campo de la Inteligencia Artificial (IA).
- Python es un lenguaje de programación avanzado universal y completamente de libre distribución (open-source).
- El matemático e informático Holandés Guido Van Rossum creo Python a partir del año 1991.
- Python es un lenguaje de programación multiparadigma (orientado a objeto, funcional, imperativo, por eventos, declarativa, entre otros).

Ventajas y desventajas de Python

Ventajas:

- Es un lenguaje interpretado y dinámico.
- Utiliza una sintaxis simple y estructuras elegante, lo cual lo hace fácil de aprender.
- Tiene una gran colección de bibliotecas desarrolladas por terceros.
- Puede invocar código escrito en otros lenguajes.

Desventajas:

- Su velocidad de ejecución es lenta.

Campos de aplicación de Python

- Python posee una vasta cantidad de bibliotecas desarrollada por tercero que le permite ser utilizado en diversos campos de acción.
- Inteligencia Artificial.
- Ciencia de Datos.
- Herramientas de compilación de sistemas.
- Desarrollo de aplicaciones.
- Rutinas de automatización de operación y mantenimiento.
- Desarrollo WEB.

Ambientes de operación de Python

- El interprete de Python puede ser bajado para variadas plataformas de sistemas operativos desde el sitio web oficial. (`https://www.python.org/`)
- Windows: `https://www.python.org/downloads/windows/`
- MacOS: `https://www.python.org/downloads/macos/`
- Linux/Unix: `https://www.python.org/downloads/source/`
- Plataforma Anaconda la cual integra multiples bibliotecas desarrolladas por terceros y es ampliamente utilizada en Inteligencia Artificial y en computación científica: `https://www.anaconda.com/`

IDE's (Integrated Development Environment)

- **Eclipse**: herramienta de desarrollo ampliamente utilizado con Java y Python. `https://www.eclipse.org/downloads/`
- **PyCharm**: Un ambiente de desarrollo dedicado a Python con funciones extremadamente poderosas y convenientes. `https://www.jetbrains.com/pycharm/download/`
- **Jupyter Notebook**: ambiente de computación interactiva basada en la Web. `https://jupyter.org/`, `https://colab.research.google.com/`
- **Visual Studio Code**: Editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. `https://code.visualstudio.com/`
- **Spyder**: es un ambiente de desarrollo integrado de libre distribución multiplataforma para programación científica en lenguaje Python. `https://www.spyder-ide.org/`

UCSC

Sintaxis básica

- Python utiliza la identación (sangrado) para definir los bloques de control. Puede ser uno o más espacios o tabuladores.
- Las bibliotecas, paquetes o módulos se pueden importar utilizando **import ...** y/o **from ... import ...**
- Para los comentarios sw utiliza **#** para comentar una línea y ’’’...’’’ o """...""" para comentar varias líneas.
- Las **palabras clave** o **keyword** son las palabras predefinidas que tienen un significado especial y que no pueden ser utilizadas para nombrar cualquier variable, función, clase, entre otros. Python 3.8 tiene **35 palabras claves**.
- Los **identificadores** son nombres de funciones, variables, clase, entre otros. No se permite que las palabras reservadas sean los nombres de los identificadores.

Sintaxis básica

Listado de palabras claves

Keyword	Descripción
as	se utiliza con **"con como"** para ejecutar dos operaciones juntas como un par
and	es un operador lógico. El operador **and** generará **True** cuando ambas condiciones sean verdaderas, de lo contrario se devolverá **False**
assert	asegura si una condición es **True**
async, await	son utilizadas para escribir código concurrente en Python
break	se utiliza para romper el flujo de control mientras se trabaja con la construcción de bucles en base a ciertas condiciones
class	define una clase en Python OOP
continue	se salta la iteración actual de una construcción de bucle en base a una condición
def	define una función en Python
del	borra cualquier referencia de un objeto
elif	representa otra condición si se construye
else	es un bloque a ejecutar cuando una condición con **if** se convierte en **False**
except	maneja las excepciones (errores de tiempo de ejecución)
finally	Una cláusula **"final"** siempre se ejecuta antes de dejar la sentencia **try**, tanto si se ha producido una excepción como si no
for	representa el ciclo iterativo o bucle **"para el"**
from	se utiliza con **import** para importar cualquier función, programa de un módulo
global	declara una variable global
if	representa la declaración **"si"**, cuyo cuerpo se ejecuta cuando la condición es **True**

Sintaxis básica

Listado de palabras claves - continuación

Keyword	Descripción
import	importa una función, clase, variable de un módulo
in	es un operador de comprobación de pertenencia o un operador de comprobación de pertenencia de un objeto secuencia en un ciclo **for**
is	comprueba la equivalencia de dos cantidades
lambda	crea la función **"lambda"** (función anónima)
nonlocal	declara variable no local cuando se trabaja con función anidada.
not	es un operador unario. Niega el valor **True** a **False** y viceversa
or	es un operador lógico que generará **True** cuando alguna de las condiciones sea verdadera, de lo contrario se devolverá **False**
pass	representa una declaración nula. **pass** se utiliza como marcador de posición
raise	plantea una excepción
return	devuelve el valor de una función
try	representa un bloque en el que hay algunas líneas de un código que pueden dar lugar a una excepción
while	representa el ciclo iterativo o bucle **"mientras"**
with	determina la configuración local de un bloque de código, conocido como **contexto**
yield	generador de retorno (genera un elemento en cada iteración) a partir de una función
None	representa un valor **null**
True	valor booleano devuelto como **verdadero** cuando se evalúa una expresión
False	valor booleano devuelto como **falso** cuando se evalúa una expresión

Sintaxis básica

Reglas de denominación de los identificadores

1. Un identificador puede tener letras (tanto en mayúsculas como en minúsculas), dígitos (0 a 9) o guión bajo (_), por ejemplo, *primer_nombre*, *mi_primer_auto* y *RutUsuario* son identificadores legales.
2. No puede usar dígitos para iniciar un identificador, por ejemplo, *1primer_nombre* es un identificador ilegal.
3. No se permite que las palabras clave sean los nombres de los identificadores.
4. No se permiten símbolos especiales como $, !, @, #, %, entre otros, en un identificador.
5. El identificador Python no tiene limitación de longitud.
6. No se permiten espacios en blanco.

Sintaxis básica

Consideraciones

1. Como Python es sensible a las mayúsculas y las minúsculas, las variables son diferentes. Por lo tanto, *primer_nombre* y *PRIMER_nombre* son dos variables diferentes.
2. Usa nombres significativos para las variables para incrementar la legibilidad de tus códigos.
3. Si tu variable contiene múltiples palabras, deben ser separadas por un guion bajo.
4. O puedes usar **camel case** para separar múltiples palabras, es decir, la primera letra de cada palabra debe estar en mayúsculas, por ejemplo, *PalabrasConMayusculas*.

UCSC

Sintaxis básica

PEP 8

- **PEP 8**: Python Enhancement Proposal 8 es una guía de estilo de como se debe escribir un código en Python, la cual ayuda a mejorar su legibilidad y elegancia. (`https://pep8.org/`)
- No es obligatorio utilizarla, pero se recomienda su aplicación.
- Ejemplo de algunas normas:
  - Siempre preferir espacios en ves de tabs.
  - Usar 4 espacios en la indentación.
  - Las líneas deben tener menos de 80 caracteres.
  - Las líneas que pasen de esta longitud deben ser divididas en dos líneas, y la línea resultante de la división debe estar indentada.
  - Las funciones deben estar declaradas en minúscula y las palabras separadas por guiones bajos **def funcion_abc()**.
  - Constantes deben estar en mayúsculas separadas por guiones bajos **NUMERO_MAXIMO = 10**.
  - Siempre coloca los **imports** al inicio del archivo.

UCSC

Sintaxis básica

Espacio de nombres y alcance

- Un **nombre** en Python es un identificador dado al objeto.
- Python es un lenguaje de programación orientado a objetos, lo que significa que todo es un objeto en Python y los nombres se usan para acceder a los objetos.
- Un **espacio de nombres** es una colección de nombres.
- Un espacio de nombres es una asignación de nombre a un objeto y se implementa como un diccionario en Python.
- Los espacios de nombres aseguran que cada nombre utilizado en un programa sea único.
- Los espacios de nombres se crean al iniciar el intérprete y se eliminan al finalizar la ejecución del programa.
- Cuando llama una función, se crea un área de nombres local que contiene todos los nombres definidos.

UCSC

Sintaxis básica

Espacio de nombres y alcance

- Cuando se crea una variable en el programa, es posible que no se pueda acceder a esa variable desde todas las partes del programa. Esto se debe al **alcance** de las variables.
- Intenta acceder a las variables desde el espacio de nombres donde no están definidas.
- Un **scope** puede definirse como un lugar desde el que puedes acceder a un espacio de nombres sin ningún prefijo.
  1. Ámbito de una función donde se tienen nombres locales.
  2. Ámbito de un módulo en el que se tienen variables globales.
  3. Ámbito de una función donde tiene nombres incorporados.
- Cuando se crea una referencia dentro de una función, se busca primero en el espacio de nombres local y luego se busca en el espacio de nombres global; si no se encuentra en ningún sitio se buscará en el espacio de nombres incorporado.

UCSC

Funciones incorporadas de Python básicas

- Lista de las funciones incorporadas en Python 3.8. en `https://docs.python.org/3.8/library/functions.html`.
- **print()**: función de salida. `print("hola mundo")` genera `hola mundo`.
- **input()**: recibe entradas del usuario.
- **del(obj)**: borra un objeto de la memoria. `a="python"; del(a)`.
- **range(start,stop,[step])**: genera una secuencia de números, indicando el *comienzo*, el *final* y el *incremento*. La secuencia termina antes del *final* definido. `range(0,20,4)`.
- **type(obj)**: devuelve el tipo del objeto. `type(print)` retorna `buildin_function_or_method`.

Funciones incorporadas de Python básicas

- **dir(obj)**: muestra los métodos incorporados y atributos de un objeto. `dir(print)`.
- **id(obj)**: muestra la dirección de memoria del objeto. `A=1; id(A)` muestra el valor `9788992`.
- **help(obj)**: despliega la información de ayuda acerca del objeto. `help(print)`.
- **len(obj)**: retorna el número de elementos de un objeto. `len([a,b,c])` retorna el valor `3`.
- **slice(obj)**: retorna una rebanada de un objeto. `A=(1,2,3,4,5,6); A[slice(2)]` retorna `(1,2)`.

Tipos de datos

- Los tipos de datos más comunes son: *números, string, listas, tuplas, diccionarios y conjuntos*.
- Los tipos de datos se pueden clasificar en: *secuenciales, no secuenciales, modificable y no modificables*.
- **Números**: *enteros, flotantes y complejos*.
- Operaciones básicas sobre números: suma (+), resta (-), multiplicación (*), división (/), potencia (**), división truncada (//), módulo o resto (%).
- Si se realiza una operación sobre números de diferentes tipos, el resultado será del tipo con mayor precisión.

Tipos de datos

String

- **String**: secuencia con múltiples caracteres. El número de caracteres determina la longitud del string.
- Un caracter es considerado como un string de longitud 1.
- Para declarar un string se puede usar 1, 2 o 3 apostrofes simples o dobles. `A='hola mundo'` o `A="hola mundo"`
- El caracter de escape (\) se puede utilizar dentro del string. `"hola \n mundo"`.
- Operador $+$ concatena dos strings. `a="hola"; b="mundo"=> a+b="holamundo"`.
- Operador $*$ repite un string *n* veces. `á'*3 => áaa'`.

Tipos de datos

String

- Lista de los métodos asociados a string en Python 3.8. en `https://docs.python.org/3.8/library/stdtypes.html#text-sequence-type-str`, y `https://www.w3schools.com/python/python_ref_string.asp`.
- Algunos métodos sobre string interesantes pueden ser los siguientes:
  - **str.split(str1)**: divide un string usado `str1` como separador.
  - **str.replace(str1,str2)**: reemplaza `str1` en el string con `str2`.
  - **str.lower()**: convierte las letras mayúsculas de un string en minúsculas.
  - **str.upper()**: convierte las letras minúsculas de un string en mayúsculas.
  - **str.join(str1)**: concatena con `str` cada elemento del string `str1`. Ejemplo, `"-".join("hola")` => `"h-o-l-a"`.

Tipos de datos

String

- Para formatear la salida de algún string se utiliza el caracter %.

Formato	Descripción
%c	Caracter y su código ASCII
%s	String
%d	Entero decimal con signo
%u	Entero decimal sin signo
%o	Entero octal sin signo
%x	Entero hexadecimal sin signo
%X	Entero hexadecimal sin signo en mayúsculas
%f	Punto flotante
%e	Punto flotante en notación científica
%E	Punto flotante en notación científica en mayúsculas
%g	Punto flotante que elige entre %e o %f
%m.n	*m* indica em ancho total y *n* los dígitos decimales
%0	Rellena con 0 a la izquierda en vez de espacios
%+	Pone el signo + para números positivos
%#	Despliega 0o ó 0x antes de un número octal o hexadecimal

Tipos de datos

Listas

- Una **lista** es una secuencia de elementos los cuales pueden ser de cualquier tipo y estos elementos pueden ser agregados o quitados de la **lista** en cualquier momento.
- Una **lista** está delimitada por un par de corchetes [ ] y los elementos de la **lista** están separados por comas [, , ,].
- Una **lista** se puede crear de dos formas:
  - `Lista = list((obj1, obj2, obj3, ...))`
  - `Lista = [obj1, obj2, obj3, ...]`
- Operador $+$ combina **listas**. `[1,2]+[3,4]` es `[1, 2, 3, 4]`.
- Operador $*$ repite los elementos de una **lista** *n* veces. `[1, 2]*3` es `[1, 2, 1, 2, 1, 2]`.

Tipos de datos

Listas

- Métodos asociados a **listas** en Python 3.8. en `https://docs.python.org/3.8/tutorial/datastructures.html`, `https://www.w3schools.com/python/python_ref_list.asp` y `https://www.python-ds.com/python-3-list-methods`.
- Algunos métodos sobre **listas** interesantes son los siguientes:
  - **list.append(obj)**: agrega un objeto al final de la **lista**.
  - **list.insert(idx,obj)**: inserta un objeto en la posición `idx` de la **lista**.
  - **list.pop([idx])**: elimina el objeto en la posición `idx` de la **lista** y retorna el elemento eliminado.
  - **list.remove(obj)**: elimina la primera instancia del objeto `obj` de la **lista**.
  - **list.index(obj)**: retorna el índice de la primera instancia del objeto `obj` de la **lista**.
  - **list.count(obj)**: retorna el número de veces que está el objeto `obj` en la **lista**.

Tipos de datos

Tuplas

- Una **tupla** es una secuencia de elementos los cuales pueden ser de cualquier tipo.
- Los objetos almacenados en una **tupla** están más seguros que en una **lista** porque no es modificable.
- Una **tupla** está delimitada por una par de paréntesis ( ) y los elementos de la **tupla** están separados por comas (, , ,).
- Una **tupla** se puede crear de tres formas:
  - `Tupla = tuple((obj1, obj2, obj3, ...))`
  - `Tupla = (obj1, obj2, obj3, ...)`
  - `Tupla = obj1, obj2, obj3, ...`
- Si una **tupla** tiene sólo un elemento cuando es creado, se debe agregar una coma (,) al final para que Python interprete de forma correcta los paréntesis finales.

Tipos de datos

Tuplas

- Las **tuplas** también cuentan con los operadores $+$ y $*$.
- Métodos asociados a **tuplas** en Python 3.8. en `https://www.w3schools.com/python/python_tuples_methods.asp`.
- Sólo dos métodos son aplicables a las **tuplas**:
  - **tuple.index(obj)**: retorna el índice de la primera instancia del objeto `obj` de la **tupla**.
  - **tuple.count(obj)**: retorna el número de veces que está el objeto `obj` en la **tupla**.

Tipos de datos

Diccionarios

- Cada elemento de un **diccionario** está compuesto por una *clave* y un *valor*, conocidos como par *clave-valor*.
- La *clave* es inmutable y única. Si en un **diccionario** se repitiera alguna *clave*, el valor de la última *clave* repetida sobreescribe los valores de las *claves* repetidas anteriores.
- Cuando un **diccionario** tiene una gran número de elementos, la velocidad de acceso a éste aumenta considerablemente comparado con el de una **lista**.
- Los elementos de un **diccionario** están delimitados por un par de llaves { } y están separados por comas {, , , }.
- Un **diccionario** se puede crear de las siguientes formas:
  - `Diccionario = {key:value, }`
  - `Diccionario = dict(key=value, )`
  - `Diccionario = dict([(key,value), ])`
- Los operadores $+$ y $*$ no son aplicables a los **diccionarios**.

Tipos de datos

Diccionarios

- Métodos asociados a **diccionarios** en Python 3.8. en `https://www.w3schools.com/python/python_ref_dictionary.asp`.
- Algunos métodos sobre **diccionarios** son los siguientes:
  - **dict.get(key)**: obtiene el valor asociado a la clave `key`.
  - **dict.items()**: retorna una lista de todas las tuplas `(key,value)` del **diccionario**.
  - **dict.keys)**: retorna una lista de todas las `keys` del **diccionario**.
  - **dict.values)**: retorna una lista de todos los `values` del **diccionario**.
  - **dict.update(dict1)**: usa `dict1` para actualizar el **diccionario**.
  - **dict.pop(key)**: borra y retorna el valor de `key`.
  - **dict.popitem()**: borra y retorna el último elemento del **diccionario**.
  - **dict.clear()**: borra todo el **diccionario**
  - **dict[key]=value**: agregar el par `{key:value}` al final del **diccionario**. Si `key` ya existe, sobrescribe su valor actual.

UCSC

Tipos de datos

Conjuntos

- Cada elemento de un **conjunto** es único y elementos duplicados son borrados.
- Los elementos de un **conjunto** están delimitados por un par de llaves { } y están separados por comas {, , , }.
- Los elementos de un **conjunto** no se pueden cambiar, ordenar, indexar ni repetir.
- Aunque en un **conjunto** los elementos no se pueden cambiar, si se puede borrar un elemento y agregar otro.
- Un **conjunto** se puede crear de dos formas diferentes:
  - `Conjunto = {obj1, obj2, ...}`
  - `Conjunto = set([obj1, obj2, ...])`
- Los operadores $+$ y $*$ no son aplicables a los **conjuntos**.

Tipos de datos

Conjuntos

- Métodos asociados a **conjuntos** en Python 3.8. en `https://www.w3schools.com/python/python_ref_set.asp`.
- Algunos métodos sobre **conjuntos** son los siguientes:
  - **set.add(obj)**: agrega un elemento, si el elemento ya existe, no se realiza la operación.
  - **set.update(obj)**: agrega un objeto que puede ser una lista, disccionario u otro. Así, múltiples objetos pueden ser agregados los cuales deben estar separados por comas (,).
  - **set.remove(obj)**: borra un elemento del **conjunto**. Si no existe, genera una excepción.
  - **set.discard(obj))**: borra un elemento del **conjunto**. Si no existe, no genera una excepción.
  - **set.clear()**: borra todo el **conjunto**.
  - **set.pop()**: borra un elemento del **conjunto** de forma aleatoria.

Tipos de datos

Copia superficial (shallow) y copia profunda (deep)

- En Python hay dos tipos de copia, una superficial o **shallow** y otra profunda o **deep**.
- Shallow copy (`copy()`) copia estructuras de datos completas y en realidad lo que se pasa de una variable a otra es la referencia a dicha estructura. Así, modificaciones realizadas en la estructura original afecta a los datos copiados en la otra variable.
- Deep copy (`deepcopy()`) copia los datos almacenados en una estructura a una nueva estructura de datos con un identificador diferente. Por lo tanto, los cambios realizados a los datos originales no afectan a la copia realizada.
- Para trabajar con estos métodos se debe importar la biblioteca como `import copy`.

Tipos de datos

Operadores

Python tiene los siguientes operadores:

- Aritméticos: `+, -, *, /, **, //, %`.
- Comparación: `==, !=, >, <, >=, <=`.
- Asignación: `=, +=, -=, /=, *=, //=, **=`.
- A nivel de bits: `&, |, ^`.
- Lógicas: `and, or, not`.
- Pertenencia: `in, not in`.
- Identidad: `is, is not`.

Más información en:

`https://www.w3schools.com/python/python_operators.asp`

UCSC

IF

- **`if`** o **`if - else`** es una estructura de control condicional que decide que bloque de código es ejecutado según la evaluación de un sentencia condicional de verdad (`True` o `False`).
- Si hubieran múltiples condiciones a evaluar se puede usar el formato: **`if - elif - else`**.

```
if condicion1:
  Bloque de sentencias 11
else:
  Bloque de sentencias 12
```

```
if condicion2:
  Bloque de sentencias 21
elif condicion3:
  Bloque de sentencias 31
else:
  Bloque de sentencias 32
```

Más información en:

`https://www.w3schools.com/python/python_conditions.asp`

FOR

- La sentencia `for` en Python es algo diferente al do los otros lenguajes en eñ sentido de que utiliza un objeto secuenciable como parámetro de entrada y lo va iterando de un elemento a la vez.
- Además se le puede agregar la sentencia `else` al final para que lo ejecute cuando el ciclo termine.
- Así, si el ciclo for no termina por una sentencia `break`, entonces se ejecuta el bloque de sentencias que están en el `else`.

```
for iter in iter_obj:
  Bloque de sentencias del ciclo for
else:
  Bloque de sentencias del else
```

Más información en:

`https://www.w3schools.com/python/python_for_loops.asp`

WHILE

- La sentencia `while` en Python ejecuta un conjunto de sentencias de forma cíclica mientras una cierta condición sea verdadera. Cuando esta condición se hace falsa, el ciclo iterativo del `while` termina.
- También se le puede agregar la sentencia `else` al final para que lo ejecute cuando la condición del ciclo `while` se hace falsa.
- Se recomienda evitar generar ciclos `while` vacíos para evitar un desperdicio de recursos de CPU. Por ejemplo, para generar un espacio de espera entre los datos de salida de un proceso.

```
while condicion:
  Bloque de sentencias cuando la condicion es verdadera
else:
  Bloque de sentencias cuando la condicion es falsa
```

Más información en:

`https://www.w3schools.com/python/python_while_loops.asp`

BREAK y CONTINUE

- Las sentencias `break` y `continue` se pueden usar si se desea interrumpir un ciclo iterativo.
- La sentencia `break` finaliza el ciclo de forma completa. Es decir, el ciclo termina y la correspondiente sentencia `else` no se ejecutaría.
- Si la sentencia `break` se usa en un ciclo anidado, el ciclo en el nivel donde se ejecuta el `break` termina y continua con la instrucción que sigue a continuación.
- La sentencia `continue` se usa para saltarse el resto de instrucciones del ciclo y empezar una nueva iteración.
- Ambas sentencias `break` y `continue` se pueden utilizar con los las estructuras de control iterativas `for` y `while`.

UCSC

Funciones

- Una función es un segmento de código organizado y que se puede reutilizar para que realiza una tarea en particular o en asociación con otras funciones o partes del código.
- Las funciones mejora la modularidad y reusabilidad del SW.
- Python utiliza la palabra clave `def` para marcar el comienzo de la función, seguido por el *nombre* de la función y sus *parámetros* que están delimitados en un par de paréntesis. Finalmente se termina la definición de la función con dos punto (`:`).
- Opcionalmente, pero muy recomendado, en las primeras líneas se agrega la documentación y descripción de la función (**DocString**) utilizando el formato para los comentarios.
- El cuerpo de la función de sebe escribir con su respectiva identación.
- Para llamar a una función, ésta siempre debe tener los paréntesis, independientemente que use parámetros o no.

Funciones

- La instrucción `return` indica, al final de la función, que ésta va a retornar resultados.
- Si la función no va a devolver algún resultado, no se pone al final la instrucción `return`, lo que daría como salida por defecto de la misma en valor `None`.
- En Python las funciones pueden retornar multiples resultados por llamada.

```
def NombreFuncion(parametros):
  ''Descripción y/o documentación''
  Cuerpo de la función
  return salida
...
algo=NombreFuncion(parametros)
...
```

Más información en:

`https://www.w3schools.com/python/python_functions.asp`

Funciones

Los parámetros de una función se pueden clasificar como:

- Parámetros requeridos: estos deben ser entregado a la función en correcto orden y número.
- Parámetros claves: Cuando se llama a la función se le asigna un valor utilizado el signo $=$.
- Parámetros por defecto: Son parámetros que asumen un valor por defecto a menos que se la asigne un valor distinto al momento de la llamada.
- Parámetros de longitud variable: la función no tiene un número predeterminado de parametros. Para capturar los parametros se utiliza el signo $*$ a un parámetro tipo lista o el signo $**$ a un parámetro tipo diccionario que los va almacenando.
- Posición de los parámetros: def func (Required arguments, Keyword arguments, Default arguments, Variable-length arguments)

Funciones

- Python tiene un tipo especial de función pequeña y anónima llamada **función lambda**.
- Una **función lambda** puede tener un número arbitrarios de parámetros, pero solo tiene una sentencia o instrucción.
- Una **función lambda** tiene su propio espacio de nombres y no puede acceder a otras variables que no sean sus parámetros o variables globales.
- Una **función lambda** se define como: `lambda parámetros : sentencia`. Por ejemplo: `x=lambda a,b:a+b; print(x(5,6))`.

Más información en:

`https://www.w3schools.com/python/python_lambda.asp`

Por consola o terminal

- Las funciones de entrada y salida estándar (teclado y pantalla) en Python están dada por `input()` y `print()`.
- la función `input()` retorna una cadena por lo que si se requiere ingresar un número se realizar una conversión.

```
nombre=input("¿Cuál es tu nombre: ?)
print("Encantado de conocerte " + nombre)
print("Encantado de conocerte ", nombre)
print(f"Encantado de conocerte {nombre}")
print("Encantado de conocerte {}".format(nombre))

edad=int(input("¿Cuál es tu edad: ?))
print(nombre + " tiene " + str(edad) + " años")
```

Más información en:

https://www.w3schools.com/python/python_user_input.asp

https://www.w3schools.com/python/python_string_formatting.asp

Por archivos

- Las funciones más comunes de entrada y salida utilizando archivos en Python están dada por `open()`, `read()`, `readline()`, `write()` y `close()`.
- La función `open()` usa al menos dos parámetros, el **nombre** del archivo y el **modo** de acceso al mismo, los cuales pueden ser **r**: abre un archivo para lectura, **a**: abre un archivo para agregar nuevos datos, **w**: abre un archivo para escritura y **x**: sólo para crear el archivo. También se puede usar **t**: para indicar que es un archivo de texto o **b**: para decir que es un archivo binario. Si se le agrega un **+** el archivo queda de lectura/escritura.
- Un ejemplo de `open` sería: `f=open("entrada.txt","rt")`.
- La función `open` retorna un objeto que tiene una serie de métodos asociados. Uno de ellos es `read` para leer el contenido completo del archivo.

Por archivos

- Un ejemplo de `read` sería: `print(f.read())`.
- Si tan solo se quiere leer los primero 5 caracteres se le agrega un parametros con el valor 5, sería: `print(f.read(5))`.
- Si se quiere leer el archivo línea a línea se utiliza el método `readline()`. Si se llega al final del archivo o **EoF** devuelve una línea en blanco.
- Un ejemplo de `readline` sería: `print(f.readline())`.
- Para escribir en un archivo se debe utilizar el método `write()` y el archivo debe haber sido abierto con los modos `"a"`, `"w"` o `"r+"`.
- Un ejemplo de `write` sería: `f=open("salida.txt","wt"); f.write("Texto de prueba")`.

Por archivos

- Siempre se recomienda cerrar el archivo cuando se deje de ocupar. Para realizar esto se invoca al método `close` como en `f.close()`.
- Otros métodos interesante para trabajar con archivos son `flush`, `tell` y `seek`.
- El método `flush` limpiar el buffer interno del archivo. Se recomienda hacerlo antes de una lectura para eliminar datos basura en ella. No requiere parámetros y no retorna valor. Ejemplo, `f.flush()`.
- El método `tell` retorna la posición actual del puntero de lectura del archivo. Ejemplo, `f.tell()` ==> `345`.
- El método `seek(off,where)` permite mover el puntero de lectura a una posición especifica. `off` es el desplazamiento y `where` es el punto de origen del desplazamiento: `0` desde el inicio, `1` desde la posición actual y `2` desde el final. Ejemplo, `f.seek(123,0)`.

Bibliotecas

- Una de las fortalezas que tiene Python es la gran base de datos de bibliotecas externas que tiene para desarrollar aplicaciones en distintas áreas de aplicación.
- Dentro del área de la Inteligencia Artificial y la Ciencia de Datos podemos nombrar las siguientes bibliotecas que son de mucha utilidad y ampliamente usadas:
- Para visualización.
  - Matplotlib.
  - Seaborn.
  - Bokeh.
- Para cálculo numérico y análisis de datos.
  - NumPy.
  - SciPy.
  - Pandas.
  - Theano.

UCSC

Bibliotecas

- Para machine learning.
  - Scikit-learn.
  - LightGBM.
- Para redes neuronales y deep learning.
  - Keras.
  - PyTorch.
  - TensorFlow.
- Para procesamiento de lenguaje natural.
  - NLTK.
  - Gensim.
  - SpaCy.
- Para bases de datos.
  - SQLAlchemy.
  - DatabaseInterface.
  - PyMySQL.

UCSC

Bibliotecas

- **Matplotlib**: utilizada para generar gráficos de calidad para publicaciones. Ofrece una variedad de gráficos tales como: series temporales, histogramas, diagramas de barras, mapas de calor, gráficos de caja y bigote entre otros. Más información en `https://matplotlib.org/stable/index.html`.
- **Seaborn**: basada en Matplotlib, esta biblioteca se especializa en la visualización de datos estadísticos. Además ofrece un alto nivel de calidad visual. Más información en `https://seaborn.pydata.org/index.html`.
- **Bokeh**: es una biblioteca para visualizar datos de forma interactiva en un navegador web. Más información en `https://docs.bokeh.org/en/latest/index.html`.
- **NumPy**: esta biblioteca ofrece estructuras de datos universales y métodos para el análisis e intercambio de datos a traves de vectores multidimensionales y matrices con capacidad para una gran cantidad de datos. Más información en `https://numpy.org/`.

Bibliotecas

- **SciPy**: proporciona rutinas numéricas eficientes fáciles de usar y opera en las mismas estructuras de datos proporcionadas por NumPy. Con SciPy se puede realizar integración numérica, optimización, interpolación, transformadas de Fourier, álgebra lineal, estadística entre otros. Más información en `https://scipy.org/`.
- **Pandas**: destaca por lo fácil y flexible que hace la manipulación de datos y el análisis de datos a traves de 2 estructuras de datos principales, las *Series* para datos en una dimensión y *DataFrame* para datos en dos dimensiones. Más información en `https://pandas.pydata.org/`.
- **Theano**: es una biblioteca de aprendizaje automático para calcular matrices multidimensionales trabajando con procesamiento de gráficos GPU y con CPU. Más información en `https://pypi.org/project/Theano/`.

Bibliotecas

- **Scikit-learn**: es una biblioteca para Machine Learning y Análisis de Datos con una gran cantidad de técnicas de aprendizaje automático para realizar aprendizaje supervisado y no supervisado. Más información en
  `https://scikit-learn.org/stable/index.html`.
- **LightGBM**: Biblioteca con modelos de árboles de machine learning diseñados para una implementación rápida y eficiente utilizando la técnica de Gradient Boosting. Más información en
  `https://lightgbm.readthedocs.io/en/latest/`.
- **Keras**: Keras es una biblioteca de alto nivel para trabajar con redes neuronales. El interfaz de Keras es mucho más fácil de usar que el de TensorFlow. Más información en
  `https://keras.io/`.

Bibliotecas

- **PyTorch**: es una biblioteca desarrollada por Facebook, que permite el cálculo numérico eficiente en CPU y GPUs. Es decir, te ofrece las capacidades de NumPy en una GPU. También se especializa en deep learning. Más información en `https://pytorch.org/`.
- **TensorFlow**: es una biblioteca desarrollada por Google, para realizar cálculos numéricos mediante diagramas de flujo de datos utilizada para deep learning y otras aplicaciones de cálculo científico. Más información en `https://www.tensorflow.org/`.
- **PyMySQL**: es una biblioteca para la interacción con bases de datos MySQL escrito completamente en Python. Más información en `https://pymysql.readthedocs.io/en/latest/`.

UCSC