24-4-2020

Grupo: 20

Autores:

* Alejo Alfredo Santi – 15955/6
* Lautaro Ezequiel Aria – 15989/8
* Fermín Moreno – 16276/2
* Jimena Games – 16174/7

Lenguajes investigados:

* Python
* Matlab

Informe de investigación

Conceptos de Lenguajes de Programación

# Introducción

A continuación, se van a detallar las características más importantes de los lenguajes Python y Matlab. Luego de mostrar los aspectos más generales de estos lenguajes, se evaluará cada uno, comparándolos entre sí, tomando como criterios de evaluación su simplicidad y legibilidad, su confiabilidad, su soporte, entre otros. Se analizará también, los aspectos sintácticos de cada lenguaje, se explicará las diferencias entre ambos en cada concepto (llamado de funciones, declaración de variables, etcétera), y habrá una mención de los operadores y estructuras de control que utiliza cada uno. Se evaluarán diferentes aspectos semánticos, para esto, se utilizará un bloque de código, tanto de Python como de Matlab, que tienen la misma funcionalidad, y se explicarán las diferencias semánticas, como funciona cada lenguaje, y se mencionarán algunos errores semánticos posibles. Por último, se analizarán las características de las variables y sus atributos utilizando el lenguaje Python.

# Python

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

# Matlab

MATLAB combina un entorno de escritorio orientado para análisis iterativo y diseño de procesos con un lenguaje de programación que expresa directamente los vectores y matrices matemáticos. Incluye el Live Editor para crear scripts que combinan código, salida y texto formateado en un documento ejecutable.

# Principales características de ambos lenguajes

Python: es un lenguaje fuertemente tipado, multiparadigma, gratis y open source, multiplataforma. Posee varios IDEs y no apunta a ningún tipo de usuario en especial, es genérico.

Matlab: es un lenguaje débilmente tipado, enfocado en operaciones con vectores y matrices, funciones, cálculo lambda. Por lo general, apunta a científicos e ingeniero, incluso la documentación está orientada para ellos. También te permite escribir directamente ecuaciones matemáticas. Tiene IDE incorporado, provee funciones y herramientas para visualizar datos en 2D y 3D, es poco portable y provee herramientas para crear aplicaciones con interfaces de usuario personalizadas. Puede llamar funciones y subrutinas escritas en C o Fortran, y se puede integrar con otros lenguajes (Java, C, C++, Python, etc). Es software propietario, por lo que hay que pagar para obtenerlo, y posee extensiones (también pagas), comparables con las librerías de Python, llamadas Toolboxes, con la diferencia que están incorporadas, no como una librería que se debe importar.

Ambos: son interpretados y de tipado dinámico. Tiene programación orientada a objetos. Son “fáciles” de leer y escribir, tienen alto nivel de abstracción y manejo de excepciones.

# Principales conceptos que atraviesan a los lenguajes de programación aplicados a los lenguajes analizados

## Criterios de evaluación

Los criterios de evaluación son los siguientes:

* Simplicidad y legibilidad: plantea que los lenguajes de programación deberían ser fáciles de escribir y leer, y por consecuencia también fáciles de enseñar/aprender. Hay ciertos conceptos que van en contra de este concepto, como pueden ser la sobrecarga de operadores, tener varias sintaxis para un mismo concepto semántico, tener distintos conceptos semánticos para la misma sintaxis, tener muchas componentes elementales o conocer subconjuntos de componentes.
* Claridad en los bindings: a los elementos de los lenguajes de programación se le pueden asignar propiedades y atributos, y esta "ligadura" del elemento con sus propiedades se puede dar en diferentes momentos, como pueden ser la escritura del programa, la compilación o la ejecución. Más allá de cuál sea el momento, está ligadura tiene que ser clara.
* Confiabilidad: está relacionada con la seguridad y se basa en dos conceptos:
  1. Chequeo de tipos: ya que al tener un problema cuanto antes se solucionen los errores de tipo antes se podrá solucionar el problema
  2. Manejo de excepciones: ya que nos permite "atrapar" las excepciones y continuar con la ejecución de un programa en caso de error
* Soporte: plantea que un lenguaje debería poder ser instalado y usado por cualquier persona, lo ideal sería que su compilador o intérprete sea de dominio público. También que se pueda utilizar en cualquier plataforma.  
  Además, debería haber varias formas para un usuario de aprender el lenguaje, libros, cursos, posteos, videos, tutoriales, etc.
* Abstracción: es la capacidad del lenguaje de manejar expresiones complejas sin tener en cuenta algunos detalles finos de la implementación.
* Ortogonalidad: para que un lenguaje sea ortogonal debería tener pocos constructores primitivos, los cuales, definiendo reglas claras de combinación, podrían llegar a construir diferentes estructuras de control o de datos. Todas las combinaciones de primitivos deberían ser posibles y con sentido.
* Eficiencia: responde a los conceptos del tiempo de ejecución, el espacio que ocupa, el esfuerzo humano, y lo optimizable que sea.

## Análisis de los lenguajes aplicando los criterios de evaluación

En cuanto a la simplicidad y legibilidad, ambos son muy simples, x = [1,2,3,4] es un vector en Matlab, y x = [1,2,3,4] en Python, incluso en Matlab con un “;” podemos transformar un vector en una matriz. Python utiliza la indentación, lo que también contribuye a la buena lectura.

Por dar otro ejemplo, un if se define con la palabra

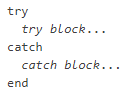
If condicion (se agrega un “:” para python)

Sentencias

end (solo para Matlab)

En reglas generales, la sintaxis es bastante limpia y fácil de aprender, no es algo super estructurado, y la indentación contribuye con la facilidad en la lectura del código. Sin embargo, ambos disponen de la sobrecarga de operadores, que si se la explota podría perjudicar el entendimiento del código, y también es verdad que tienen varías sintaxis para definir lo mismo, por ejemplo x = list() es igual a x = [] en Python, o en Matlab x = [1 2 3 4] y x = [1,2,3,4], lo que también atenta contra el principio que estamos tratando.

En cuanto a la confiabilidad, ambos tienen manejo de excepciones:

En Matlab se usa try y catch:

En Python try except y se puede especificar el error

try:

x = int(input("Por favor ingrese un número: "))

except ValueError:

print("Oops! No era válido. Intente nuevamente...")

Y en cuanto al chequeo de tipos, ambos tienen tipado dinámico, por lo que en proyectos grandes puede ser un problema, aunque cabe aclarar que se respetan ciertas reglas básicas, como, en Python, no poder sumar un número con un string.

En cuanto al soporte, Python es open source y gratis, lo que lo hace fácilmente accesible y tiene una gran documentación, con una comunidad que brinda gran cantidad de cursos y videos. En cuanto a Matlab se debe pagar, es código propietario, y la documentación que se puede encontrar es escasa.

En lo que la abstracción se refiere, ambos son lenguajes de alto nivel capaces de ignorar detalles finos de implementación en muchos casos. Matlab tiene muchísimas funciones útiles para manejar elementos matemáticos e incluso para graficarlos, de manera muy simple. Python tampoco se queda atrás, tiene muchas funciones que ofrecen un manejo simple de datos, como **map** o **sum** por ejemplo, y estructuras con altos niveles de abstracción como diccionarios o arreglos de tamaño flexible (**listas**).

Se considera eficiente a los dos programas en ámbitos científicos, ambos suelen usarse en ámbitos relacionados con Big Data o Machine Learning. En especial, Python tiene un mayor rendimiento, específicamente en las iteraciones, que en Matlab no son las mejores. También señalar que el consumo de RAM de Matlab suele ser elevado por el tipo de operaciones que realiza.

## Sintaxis

Establece reglas que definen cómo deben combinarse los componentes básicos, llamadas “word” (combinación de letras y símbolos), para formar sentencias y programas.

## Análisis sintáctico de los lenguajes y su comparación

|  |  |  |
| --- | --- | --- |
| Concepto | Python | Matlab |
| Comentario (por línea) | #texto del comentario  (exceptuando la línea  # -\*- coding: utf-8 -\*-  que define la codificación | %texto del comentario |
| Documentar función o comentar varias líneas. | “”” texto descriptivo  texto descriptivo  texto descriptivo “”” | % texto descriptivo  % texto descriptivo  % texto descriptivo  % texto descriptivo  % texto descriptivo |
| Definición de funciones | def nombreFunción (arg1, arg2):  sentencias  return (salida1, salida2)  El return es opcional  Para funciones anónimas:  funcionAnonima = lambda var: print(‘hola’) | function [salida1, salida2] = myfun(arg1, arg2)  sentencias  end  Para funciones anónimas:  funcionAnonima = @(var) fprintf(‘hola’); |
| IF - Switch | if condición :  sentencias  elif otra\_condición:  sentencias  else:  sentencias | if condición  sentencias(la indentación no es obligatoria)  elseif otra\_condición  sentencias  else  sentencias  end  switch var  case valor1  sentencias  case valor2  sentencias  otherwise  sentencias  end |
| Operadores lógicos y de comparación | == (igualdad)  != (diferencia)  or (or)  and (and)  not (not) | == (igualdad)  ~= (diferencia)  or(a,b) o | (or)  and(a,b) o & (and)  ~ o not(a) (not) |

|  |  |  |
| --- | --- | --- |
| Operadores numéricos | + (adición)  - (resta)  \* (multiplicación)  / (división)  % (módulo)  \*\* (exponencial) | + (adición)  - (resta)  \* (multiplicación)  / (división)  mod (a,b) (módulo)  ^ (exponencial) |
| Definición de vectores o listas | arr = [10 20 30]  (crea una lista de 3 elementos)  arr[0]  (accede al primer elemento)  arr[-1]  (accede al último elemento) | arr = [10 20 30]  (crea un vector de 3 elementos)  arr(1)  (accede al primer elemento)  arr(end)  (accede al último elemento) |
| Longitud de una lista o vector | len(lista) | length(vector) |
| Imprimir en consola | print(valor) | disp(valor), fprintf(valor) y sprintf(valor)  Son 3 opciones diferentes con diferencias en el formato que devuelve |
| Definición de una matriz | var = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] | var = [1 2 3; 4 5 6; 7 8 9] |

Como breve conclusión, ambas sintaxis son bastante legibles y cómodas de utilizar, son muy similares entre sí, con funciones y operadores intuitivos. Vale la pena destacar que el uso de indentación para la definición de bloques en Python, si bien es cómoda para ver, queda todo bien ordenado y estructurado, puedo representar un problema con programas muy largos donde un fallo de tipeo con un espacio en blanco puede generar errores.

## Semántica

La semántica describe el significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático. Se divide en dos, la semántica estática, la cual se encarga mayormente de chequear el cumplimiento de las formas válidas, y la semántica dinámica, la cual evalúa el código en ejecución basándose en ciertas reglas.

## Análisis semántico de los lenguajes y su comparación

Para comparar la semántica de los dos lenguajes, decidimos crear un programa de prueba con la siguiente funcionalidad:

Se leen 9 números de consola, creamos una matriz de 3x3, calculamos los valores totales de cada fila.

Pasamos el valor de la fila 0 a una función que evalúa si el número es par, si es par nos devuelve True y, si efectivamente es par, llamamos a otra función que decrementa el número de a 5 unidades hasta que quede menor que 50 y lo retorna.

Códigos utilizados:

CÓDIGO PYTHON ([anexo 1](#_Anexo_1:_código))

CÓDIGO MATLAB ([anexo 2](#_Anexo_2:_código))

Análisis comparativo de la semántica:

En las primeras líneas se puede ver la definición de las funciones, si bien ambas son muy parecidas, la definición de Matlab determina la variable de retorno (en el código nombrada como res), mientras que en Python los valores se retornan con return, no en variables. También Matlab utiliza el end para determinar el fin de la función, mientras que Python se basa únicamente en la indentación.

Ambas funciones utilizan el concepto de DocString, en donde, con un comentario, se le puede brindar una descripción a la función.

En la función esPar, ambos lenguajes se manejan de manera similar, pero Python, para obtener el módulo, utiliza el operador “%”, mientras que Matlab utiliza una función llamada mod. Ambos esperan un operando numérico para aplicarle mod.

En la función de decrementar, además de las diferencias con el return o end antes nombradas, la única diferencia que se presenta es el print (python) y el disp (matlab), ya que el disp viene acompañado de una función strcat, que es la encargada de concatenar el string con el número, el cuál previamente fue convertido a string. Luego el resto del funcionamiento es similar, ambas funciones entran en loop evaluando la sentencia que acompaña al **while**, y se va decrementando la variable que contiene el número hasta que la sentencia dé falsa. Luego, retornan el número, cada uno a su manera.

Python no te permite acceder al índice de una variable que no esté definida como indexable, por lo que antes de asignarle los números de la matriz la creamos vacía. Con Matlab no tenemos este inconveniente.

Ambas matrices se inicializan de la misma manera, haciendo **for**s anidados. En los dos lenguajes, al for hay que pasarle una variable que funciona como índice, la cual en cada iteración va tomando los valores definidos en el rango. El rango en Python se puede crear con la función **range(x,y)** y en Matlab lo mismo se realiza con x:y, los rangos están definidos como vectores con los números enteros del rango.

En cuanto a la asignación, en Python agregamos el valor al final de la lista con **append**, en Matlab accedemos al índice de la matriz directamente y asignamos los valores.

En el input, ambos leen del archivo stdin la entrada de consola, y al ingresar el número Python lo guarda como string y Matlab como número.

Para sumar los totales por fila, en Python accedemos a la lista que se corresponde con la fila y llamamos a la función **sum()**, que suma los valores de esa lista. En Matlab usamos la función **sum()** pero actúa diferente. Debemos pasarle la matriz y su dimensión, y luego devolverá un vector con la suma de cada fila.

La invocación de la función es igual, ambas guardan en una variable el retorno de las mismas. Recalcar que el parámetro esperado es un número, por las operaciones que las mismas realizan.

Los if del final funcionan igual en ambos, se evalúa la expresión y en caso de ser verdadera se decrementa el número y se lo muestra en consola, y en caso de ser falsa, se muestra un mensaje.

Otro caso a señalar, es que la igualación de una variable con la matriz, por ejemplo:

A = matriz

En Python crea A como un puntero que apunta a la variable matriz, mientras que en Matlab lo que sucede es que se crea A como una matriz con los mismos valores que se encuentran en la variable matriz.

## Semántica y chequeos de cada lenguaje

Hablando más específicamente de los errores que se identifican en semántica estática (aunque Python y Matlab sean interpretados siguen teniendo chequeos de formas válidas), podemos nombrar:

Algunos de Python:

* Chequeo de tipos básico. Un ejemplo puede ser la concatenación, Python no deja concatenar variables del tipo string y variables numéricas, por lo que primero se pasa las variables numéricas a string con la función **str().**
* Chequeo de apertura de bloques. Cada bloque comienza con un “:” y una mayor indentación.
* Chequeo de indentación. Si Python detecta que la indentación no es correcta no nos permite ejecutar el código.
* Chequeo de funciones o variables inexistentes. Python no deja utilizar una función o variable antes de definirla.
* Chequeo de cantidad de parámetros. Python chequea que la cantidad de parámetros que se envían sea la misma que la función espera, por lo que si no coinciden da error.
* Chequeo de balance de paréntesis, corchetes o llaves. Siempre se chequea que la cantidad de apertura sea la misma que de cierre, y haya coherencia entre los mismos.
* Chequeo de librerías inexistentes. Cuando se intenta importar una librería con **import**.
* Chequeo de archivos inexistentes. Cuando se intenta abrir o acceder a un archivo.

Hay más chequeos de errores, y algunos pertenecen a la semántica dinámica, un caso puede ser:

* Chequeo de variables indentables. Python debe comprobar que las variables a las que se intenta acceder con **[posición]**, como la del for o las listas, sean indexables, y que la posición a la que se busca acceder sea válida.
* Chequeo de atributos o métodos existentes. Cuando se intenta mandar un mensaje a un objeto para acceder a una variable o llamar a un método del mismo.

Algunos de Matlab (similares a los de Python):

* Chequeo de tipos básico. Matlab también tiene chequeo de tipos, por ejemplo, al hacer funciones como **mod**, en donde los parámetros no pueden ser de tipo string. Sin embargo, Matlab suele convertir los strings a números según su código ASCII, por lo que los errores de tipo en cuanto a strings son menos frecuentes de los que parecen.
* Chequeo de funciones o variables inexistentes. Nos da un error de variable o función no definida.
* Chequeo de operaciones entre matrices. Matlab nos permite realizar varias operaciones con matrices, y cada operación debe cumplir con diferentes parámetros. Un ejemplo es tratar de igualar matrices de diferentes dimensiones, lo que genera un error.
* Chequeo de balance de paréntesis, corchetes o llaves. Siempre se chequea que la cantidad de apertura sea la misma que de cierre, y haya coherencia entre los mismos.
* Chequeo de indentación. Si Python detecta que la indentación no es correcta no nos permite ejecutar el código.

Algunos errores dinámicos de Matlab:

* Chequeo de librerías o archivos inexistentes. Por ejemplo, la función loadlibrary('*nombreLibreria*','*archivoHeader*') busca una librería y un archivo, y si no encuentra alguno de los dos nos dará un error.
* Chequeo de variables indentables y dimensiones. Matlab hace chequeos para corroborar que la variable a la que se intenta acceder sea indexable, que la posición sea válida, y que coincidan las dimensiones.
* Chequeo de atributos para clases o estructuras. Al llamar a un método de un objeto, clase o estructura particular, comprueba que el mismo exista.

## Semántica

Una variable es, en pocas palabras, un espacio de memoria reservado para almacenar un valor determinado, dependiendo del lenguaje de programación en el cual se trabaja.

Vamos a basar la explicación de las variables en Python 3:

Python, en cuanto al alcance de sus variables, tiene un alcance estático, esto quiere decir que se define según la estructura léxica del programa. Cuando Python busca una variable lo hace primero en el bloque en el que se encuentra, y si no la encuentra, va al bloque que la contiene, sin importar el orden de ejecución. Tiene 2 palabras claves para el manejo de variables, nonlocal, que va a buscar una variable al bloque anterior, y global, que va a buscar la variable al código “principal” o el bloque global.

Python utiliza una técnica llamada espacio de nombres, en donde a cada variable definida le asigna su objeto correspondiente. Hay un espacio de nombres local a cada función y uno global.

Código de ejemplo ([anexo 3](#_Anexo_3:_código))

Como Python es interpretado, el tiempo de vida y el alcance de las funciones y variables no es representable con una tabla, por lo que se explicará su funcionamiento en forma de texto. Tiene dos particularidades:

El tiempo de vida de un elemento no mutable (tipos primitivos en general) se va modificando, ya que cada vez que se le asigna un nuevo valor se crea un objeto que se aloca en una nueva posición de memoria y se apunta hacia ese.

El alcance de las funciones o variables es relativo a la ejecución. Uno pensaría que es desde su declaración hacia abajo, ya que no podemos acceder a las mismas antes de declararlas, pero puede darse el caso en que se acceda desde antes de su declaración mediante una función que se ejecute después. Es decir, si yo defino funcion1 y luego funcion2, y en el código, luego de ambas declaraciones, invoco la funcion1, esta podrá llamar a la function2 aunque esté situada antes en el código, ya que la declaración de la funcion2 ya se hizo. Como el alcance de las variables y funciones depende de la ejecución, no solo del lugar de la declaración, se irá detallando en el texto el funcionamiento y acceso de las mismas.

Se interpreta línea por línea, se detalla a continuación lo que ocurre:

Primero se lee function1, y se agrega function1 al espacio de nombres global, por lo que cualquier referencia que se haga en las líneas siguientes cargará la función a memoria. Como todavía no fue llamada, el tiempo de vida no comenzó.

Sigue interpretando después de toda la función (no lee el contenido de la misma), y se encuentra con function2, y es el mismo procedimiento. Guarda function2 en el espacio de nombres global y sigue interpretando después de la función. De ahora en adelante la function2 ya puede ser invocada, su alcance comenzó se podría decir, ya que está en el espacio de nombres, pero su tiempo de vida todavía no empezó.

Se lee la variable “x” y se guarda en el espacio de nombres global. Al nombre o etiqueta “x” se le asigna o apunta a un objeto con valor 10. Comienza su alcance y tiempo de vida.

Se lee la variable “y” y se guarda en el espacio de nombres global. Al nombre o etiqueta “y” se le asigna o apunta a un objeto con el string ‘hola’. Comienza su alcance y tiempo de vida.

Se lee la variable “lista” y se guarda en el espacio de nombres global. Al nombre o etiqueta “lista” se le asigna o apunta a un objeto de tipo list vacío. Comienza su alcance y tiempo de vida.

Se agrega la variable “x” como un elemento de la lista. Como la lista es un elemento mutable, su tiempo de vida (u objeto referenciado por la etiqueta) no cambia.

Se agrega la variable “y” como un elemento de la lista.

Se llama a fuction1. El tiempo de vida de function1 comienza.

Se lee la variable “x” y se la agrega al espacio de nombres local a la función function1. Se le asigna un objeto con valor 20. Hasta que termine la función, esta “x” con valor 20 es alcanzable. También comienza su tiempo de vida.

Se lee la definición de function3 y se la agrega al espacio de nombres local a la función function1. Por lo que ya puede ser referenciada en el contexto de function1.

Se continúa interpretando, se imprime x, la cual se encuentra en el espacio de nombres local a la función function1, con el objeto de valor 20.

Se invoca function2. Aunque la declaración de function2 esté debajo de la línea que se invoca, function2 ya se encuentra cargado en el espacio de nombres global, por lo que puede ser referenciada.

Se va a function2, se podría decir que su tiempo de vida comienza. Se realiza el print de la variable “x”, la cual se encuentra en el espacio de nombres global (con valor 10).

Termina function2, su tiempo de vida termina y se sigue interpretando en function1.

Se invoca function3. Su tiempo de vida comienza.

Se aclara que la variable “x” que se va a utilizar es la que está en el espacio de nombres global.

Se incrementa en el valor de la variable “x”, lo que genera un nuevo objeto de valor 11, al cual ahora va a apuntar la etiqueta que se encuentra en el espacio de nombres global. Esto genera que el tiempo de vida de “x” “vuelva a comenzar”, ya que es un objeto nuevo recién alocado.

Se realiza el print de la “x” antes mencionada, que tiene valor 11. Termina function3 y se termina su tiempo de vida.

Se termina function1 y finaliza su tiempo de vida.

Termina el programa.

# Bibliografía

The MathWorks, Inc.: la.mathworks.com

Último acceso 22/4/2020

Real Python: [realpython.com](https://realpython.com/matlab-vs-python/)

Último acceso 23/4/2020

John W.Eaton: octave-online.net

Último acceso 22/4/2020

Elena Álvarez Sáiz, Dpto. Matemática Aplicada y C. Computación, Universidad de Cantabria: Matemáticas 1 MATLAB Comandos y ejemplos

(https://ocw.unican.es/pluginfile.php/2021/course/section/2379/comandos-ejemplo-matlab.pdf)

Último acceso 4/4/2020

Mª Cristina Casado Fernández, Servicios Informáticos U.C.M Apoyo a Investigación y Docencia: Manual Básico de Matlab

(http://webs.ucm.es/centros/cont/descargas/documento11541.pdf)

Último acceso 18/4/2020

Departamento de Ingeniería Matemática, Facultad de Ciencias Físicas y Matemáticas, Universidad de Concepción: Introducción a Matlab II (2018)

(ftp://ftp.ing-mat.udec.cl/pub/ing-mat/asignaturas/521230/laboratorios/2018-1/lab02-521230-2018.pdf)

Último acceso 4/4/2020

Refsnes Data : [www.w3schools.com](https://www.w3schools.com/)

Último acceso 23/4/2020

Alumno de AMU(Aligarh Muslim University), Mr. Mohtashim: <https://www.tutorialspoint.com/codingground.html>

Último acceso 22/4/2020

Richard K. Herz y sus estudiantes en la Universidad de California: [reactorlab.net](http://reactorlab.net/resources-folder/matlab/P_to_M.html)

Último acceso 18/4/2020

# Anexos

### Anexo 1: código de ejemplo de Python

### Anexo 2: código de ejemplo de Matlab

Imprime:

### Anexo 3: código de ejemplo para el análisis de variables