

INSTITUTO TECNOLÓGICO SUPERIOR DE CHICONTEPEC

INGENIERÍA EN SISTEMAS COMPUTACIONALES



NOMBRE DE LA MATERIA:

MÉTODOS NUMÉRICOS.

NOMBRE DEL TEMA:

TIPOS DE DATOS Y ESTRUCTURA DE CONTROL (PYTHON).

NOMBRE DEL ALUMNO:

MIGUEL ANGEL MARTINEZ MARTINEZ.

NOMBRE DEL DOCENTE:

ING. EFREN FLORES CRUZ.

Índice

Introducción.....	3
Python: Números y operaciones aritméticas	4
Cadenas de texto	9
Variables	14
Booleanos	26
Secuencias: Tupla, Rangos, Lista.....	31
Estructura de control: Sentencias condicionales (if, elif, else).	49
Iteraciones (bucle for)	49
Iteraciones (bucle while).....	50
Conclusión.....	51
Bibliografía.....	52



Introducción

Python es un lenguaje de programación multiparadigma ya que soporta programación orientado a objetos, una programación imperativa y en menor medida, de tal forma que lo hace una programación fundamental.

Python es un lenguaje de programación de Scripting de plataforma y además de ser orientada a objetos, preparado para realizar cualquier tipo de programación, como puede ser aplicaciones para Windows y no cabe mencionar servidores de red o incluso páginas web. Es un lenguaje interpretado, lo que significa que no necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventaja, como la rapidez de desarrollo.



Python: Números y operaciones aritméticas

Números y operaciones aritméticas elementales

Enteros y Decimales

Python distingue entre números enteros y decimales. Al escribir un número decimal, el separador entre la parte entera y la parte decimal es un punto.

>>> 3

3

>>> 4.5

4.5

Nota: Si se escribe una coma como separador entre la parte entera y la decimal, Python no la entiende como separador, sino como una pareja de números (corralamente, lo entiende como una tupla de dos elementos, un tipo de dato).

>>> 3,5

(3,5)

Si se escribe un número decimal con parte decimal 0, Python considera el número como número decimal

>>> 3.0

3.0

Se puede escribir un número decimal sin parte entera, pero lo habitual es escribir siempre la parte entera.



• Números complejos

Python puede hacer cálculos con números complejos. La parte imaginaria se toma acompaña de la letra "j".

```
>>> 1 + 1j + 2 + 3j  
(3+4j)  
>>> (1+1j) * 1j  
(-1+1j)
```

La letra "j" debe ir acompañada siempre de un número y unidad en el caso de los números complejos.

```
>>> 1 + j
```

Traceback (most recent call last):

File "<pyshell#25>", line 1, in <module>

1 + j

NameError: name 'j' is not defined

```
>>> 1 + 1j
```

SyntaxError: invalid syntax

```
>>> 1 + 1j
```

(-1+1j)

El resultado de una operación en la que intervienen números complejos es un número complejo, aunque el resultado no tenga parte imaginaria.

```
>>> 1j * 1j
```

(-1+0j)



• *Jar cuatro operaciones básicas*

Las cuatro operaciones aritméticas básicas son la Suma (+), la resta (-), la multiplicación (*) y la división (/).

Al hacer operaciones en las que intervienen números enteros y decimales, el resultado siempre es decimal. En el caso de que el resultado no tenga parte decimal, python escribe 0 como parte decimal para indicar que el resultado es decimal:

>>> $4.5 * 3$

13.5

>>> $4.5 * 2$

9.0

Al sumar, restar o multiplicar números enteros, el resultado es entero:

>>> $1 + 2$

3

>>> $3 - 4$

-1

>>> $5 * 6$

30

Al dividir números enteros, el resultado es siempre decimal, aunque sea un número entero. Cuando python escribe un número decimal, lo escribe con parte decimal, aunque sea nula:

>>> $9 / 2$

4.5

>>> $9 / 3$

3.0



Dividir por cero genera un error

Cuando en una formula aparece en varias operaciones, Python las evalúa aplicando las reglas usuales de precedencia de las operaciones (primero multiplicación y división, después suma y resta).

>>> 1 + 2 * 3

~~7~~

>>> 10 - 4 * 2

~~2~~

En caso de querer que las operaciones se realicen en otro orden, se deben utilizar paréntesis.

>>> (5+8)/(7-2)

2.6



Al realizar operaciones con decimales, los resultados pueden representar errores de redondeo.

>>> 100/3

33.333333333333336

Este error se debe a que Python almacena los números decimales en binario y pasar de decimal a binario provoca errores de redondeo. Es un error que sucede con todos los lenguajes de programación. Si necesitas una precisión absoluta, debemos utilizar bibliotecas específicas.



Debido a los errores de redondeo, las operaciones que debieran dar el mismo error resultado pueden dar resultados diferentes.

$$>>> 4 * 3 / 5$$

$$2.4$$

$$>>> 4 * 3 / 5$$

$$2.4000000000000004$$

Se pueden escribir sumas y restas seguidas, pero no se recomienda hacerlo porque no es una rutina habitual.

$$>>> 4 3 + - + 4$$

$$-1$$

$$>>> 3 - + - + 4$$

$$7$$

Lo que no se puede hacer es escribir multiplicaciones y divisiones seguidas

$$>>> 3 * 1 / 4$$

SyntaxError: invalid syntax

$$>>>$$



Cadenas de texto

Cadenas de texto

Una cadena es una secuencia imprimible de caracte^rs Unida^d por comillas

>>> print('Esto es una cadena')

Esto es una cadena

>>> print("Esto es una cadena")

Esto es una cadena

La función print(), que comenta con mas detalle en la lección Salida por pantalla, muestra en pantalla el contenido de la cadena, pero no los comillas.

En otros lenguajes las comillas dobles y las comillas simples son completamente equivalentes, pero en otros lenguajes si programas bien no lo son

⚠ En otros lenguajes de programación, por ejemplo PHP, en las cadenas delimitadas con comillas dobles las variables se sustituyen por su valor y se pueden utilizar caracteres especiales, pero en las cadenas delimitadas con comillas simples, no.

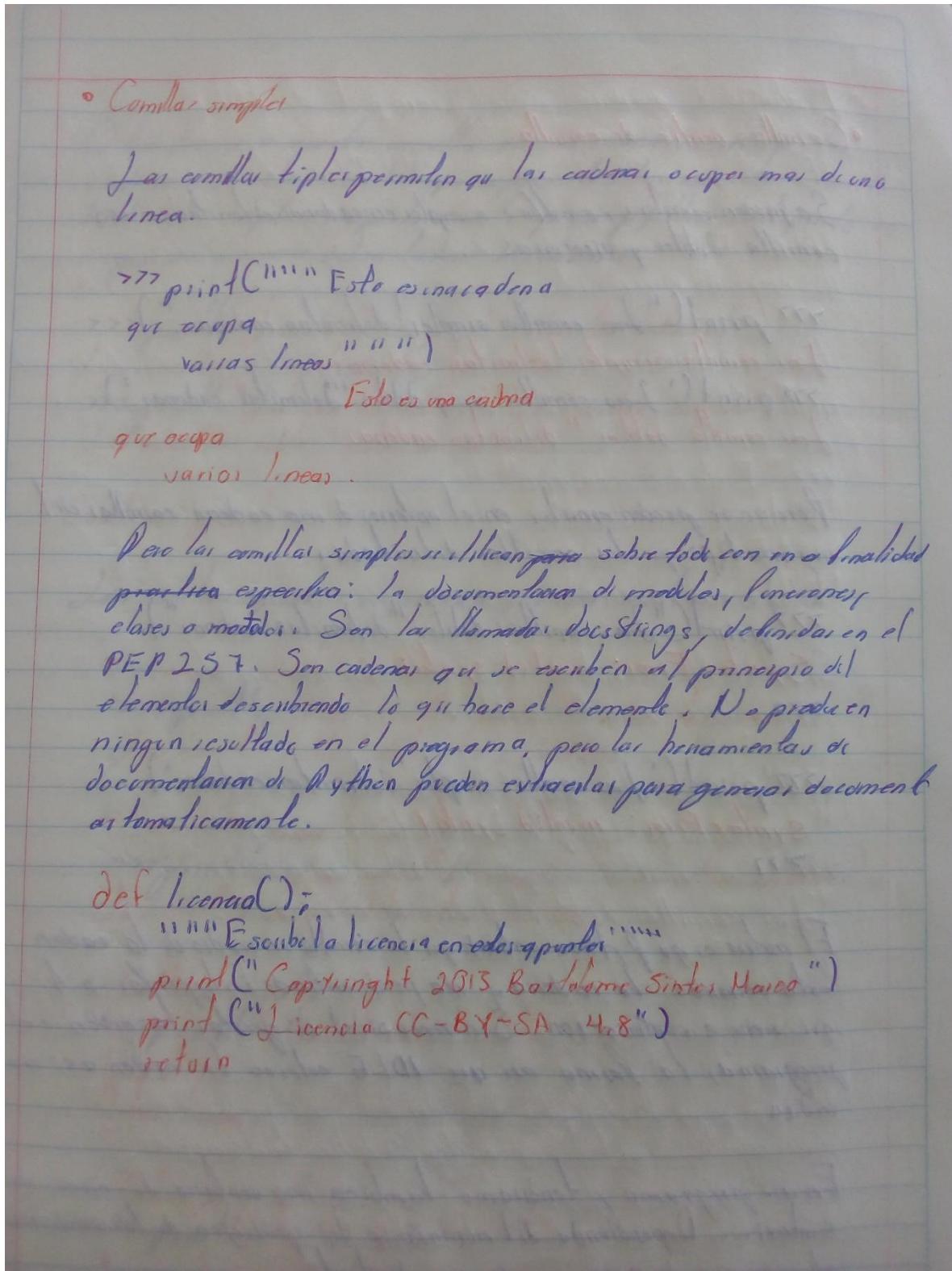
Las cadenas se deben concatenar con las comillas con las que se abrieron, de lo contrario estaremos cometiendo en errores de sintaxis

>>> print("Esto es una cadena")

SyntaxError: EOL while scanning string literal

>>>

En un programa, al intentar ejecutar se obtiene una colección de errores





• Comillas dentro de comillas

Se pueden escribir comillas simples en cadenas delimitadas con comilla dobles y viceversa:

>>> print("Las comillas simples 'delimitan cadenas.")

Las comillas simples 'delimitan cadenas'

>>> print('Las comillas "simples" delimitan cadenas.')

Las comillas dobles "delimitan cadenas"

Pero no se pueden escribir en el interior de una cadena comillas del mismo tipo que las comillas delimitadas:

>>> print("Las comillas dobles ""delimitan cadenas")

SyntaxError: invalid syntax

>>>

>>> print('Las comillas simples 'delimitan cadenas')

SyntaxError: invalid syntax

>>>

El motivo es que Python entiende que la comilla en medio de la cadena esla cerrando la cadena y no puede entenderlo si la sigue o le viene a continuación. Sin necesidad de explicar la orden a el programa, la forma en que IDLE indica la orden así no lo indica

En un programa, tendremos también una ventana de error de sintaxis. Dependiendo del número y la posición de las cadenas, el mensaje de error puede ser distinto



Otra forma de escribir comillas en una cadena es utilizar las cadenas especiales " \" " que representan los caracteres comillas dobles y simples respectivamente y que pueden intercambiarse en algún caso como alternativa de cadena.

>>> print(" Las comillas simples \" delimitan cadenas. ")

Las comillas simples " delimitan cadenas.

>>> print(" Las comillas dobles \"\" delimitan cadenas. ")

Las comillas dobles " delimitan cadenas.

*Caracteres especiales:

Los caracteres especiales emplean para una contrabarra (\)

- comilla doble: "

>>> print(" Las comillas dobles \"\" delimitan cadenas. ")

Las comillas dobles " delimitan cadenas.

- comillas simples: "

>>> print(" Las comillas simples \"\" delimitan cadenas. ")

Las comillas simples " delimitan cadenas.

- Salto de linea: \n

>>> print(" Una linea \n Otra linea")

Una linea

Otra linea

- Tabulado: \t

>>> print(" 1\t2\t3")

1 2 3



- Evaluación de cadenas en el prompt 1 de IDLE

En el prompt de IDLE se proporcionan cadenas sencillas, sin necesidad de escribir la función print(). Al pulsar Intro, IDLE evalúa la cadena y escribe el resultado, como si se estuviera ejecutando una función anónima.

En la mayoría de los casos IDLE evalúa el resultado con comillas simples, para indicar si es un tipo de cadena.

>>> "Esto es una cadena"

'Esto es una cadena'

>>> 'Esto es una cadena'

'Esto es una cadena'

Si el resultado contiene únicamente comillas simples, IDLE lo escribe entre comillas dobles.

>>> "Las comillas simples delimitan cadenas."

"Las comillas simples delimitan cadenas."

>>> 'Las comillas simples \\" delimitan cadenas.'

"Las comillas dobles \" delimitan cadenas."

Pero si aparecen ambas comillas, IDLE escribe comillas simples y las comillas simples se muestran como cadenas especiales.

>>> 'Las comillas simples \' y dobles \" delimitan cadenas.'

'Las comillas simples \' y dobles \" delimitan cadenas.'

>>> "Las comillas simples \' y dobles \" delimitan cadenas."

'Las comillas simples \' y dobles \" delimitan cadenas.'



Variables

Variables

¿Qué es una variable?

El concepto de "Variable" proviene de las Matemáticas, una variable es un símbolo que comparte de una expresión o de una fórmula. Normalmente las variables se representan mediante letras del alfabeto latino (x, y, z, n, i, j, \dots). Dependiendo del contexto, la variable significará cosas distintas. Por ejemplo

- En el caso de álgebra, una variable representa una cantidad desconocida que se relaciona con otra, y en algunos casos podemos averiguar el considerando por ejemplo la ecuación:

$$x + 3 = 5$$

En este caso, la variable x representa una cantidad desconocida por lo que se sabe si se suma 3 se obtiene 5. Resolviendo la ecuación, obtendremos inmediatamente que la variable x estaba representando realmente el número 2.

- En el caso de análisis matemático, una variable no representa una cantidad determinada, sino que representa un conjunto de valores. Consideremos por ejemplo la creación de la ecuación

$$y = x + 1$$

En este caso, la variable x no representa ningún valor concreto, sino que puede tomar cualquier valor número positivo o negativo. Para cada valor de x podemos calcular el valor correspondiente de la variable y . Si entendemos x e y como las coordenadas en un plano y dibujamos varias puntas, podemos ver que todas las puntas se encuentran situadas en una misma recta.



• Variables en programación

En programación también existe el concepto "Variable", parecido pero no idéntico al concepto matemático.

En programación las variables están asociadas a valores concretos. Además, cada lenguaje de programación tiene su forma de implementar el concepto de variables, por lo si explica a continuación la validez para muchos lenguajes de programación; aunque otros lenguajes o programaciones permiten otras posibilidades.

- En muchos lenguajes de programación, una variable se può considerar como una especie de caja en la que se puede guardar un valor (por ejemplo, un valor numérico). Esta caja suele corresponder a una posición de memoria en la memoria del ordenador.
- Las variables se procesan también mediante letras o palabras completas: x, y, a, b, nombre, apellido, edad, etc.
- Cuando en estos lenguajes de programación escribimos la instrucción ..
 $a = 2$
lo que estamos haciendo al programa es que guarda el valor 2 en una "caja" y que la caja se llame a.

En estos lenguajes, el símbolo igualdad (=) hay que entenderlo como una asignación, no como una igualdad matemática. Al escribir una igualdad, lo estamos pidiendo al programa que calcule lo que hay a la derecha y que guarde la variable que hay a la izquierda de la igualdad.



En ese lenguaje, no estan permitido escribir:

$$2 = a$$

porque 2 no es un nombre valido. Tampoco estan permitido escribir,

$$x + 3 = 5$$

porque en el lado izquierdo no puedo aparecer operadores.

Una vez que los guardado un valor en una variable, podemos hacer referencia a lo largo del programa. Describo, el siguiente programa.

$$a = 2$$

$$b = 3$$

$$c = a + b$$

- guarda el valor 2 en la variable a
- guarda el valor 3 en la variable b
- coge los valores guardados en la variables a y b (2 y 3), los suma ($2 + 3$) y el resultado (5) lo guarda en la variable c.

La informacion se guarda en variables de tipo:

- numeros enteros, decimales, imaginarios, en notacion científica, con precision arbitraria, en base decimal o en otras bases, etc.).
- cadena de texto (una serie de letras o mas letras, del juego de caracteres ASCII occidental o del juego de caracteres Unicode, etc.).
- conjunto de numero o texto (matrices, listas, tuplas, etc.).
- estructura mas complicada (punteros, diccionarios, etc.).

Variables en Python

En algunos lenguajes de programación, las variables se pueden considerar como "cajas" en las que se guardan los datos, pero en Python se llaman "etiquetas" que permiten hacer referencia a los datos (que se guardan en mas "cajas" llamadas objetos).



Python es un lenguaje de programación orientada a objetos y su modelo de datos también está basado en objetos.

Para cada dato que aparece en un programa, Python crea un objeto que contiene. Cada objeto tiene:

- un identificador (un número entero, decimal, cadena de caracteres, etc.). El identificador permite a Python referirse al objeto sin ambigüedades.
- un tipo de dato (enteros, decimal, cadena de caracteres, etc.). El tipo de dato permite saber a Python qué operaciones puede hacer con el dato.
- un valor (el propio dato)

Así las variables en Python no guardan los datos, sino que son simples nombres para poder hacer la referencia a ese objeto. Si escribimos la instrucción:

$a = 2$

Python:

- crea el objeto "2". Este objeto tiene un identificador único que se asigna en el momento de la creación y se conserva a lo largo del programa. En este caso, el objeto creado es tipo número entero y guarda el valor 2.
- asocia el nombre a al objeto número entero 2 creado.

Aquí, al descomponer la instrucción anterior no abría que decir "la variable a almacena el número entero 2"; sino abría que decir "podemos llamar a al objeto número entero 2". La variable a es como una etiqueta que nos permite hacer referencia al objeto "2".



De esta forma, Python se distingue entre objetos mutables y objetos inmutables.

- Los objetos inmutables son objetos que no se pueden modificar. Por ejemplo, los números, las cadenas y las tuplas son objetos inmutables.
- Los objetos mutables son objetos que se pueden modificar. Por ejemplo, los listas y los diccionarios son objetos mutables.

• Definición:

Las variables en Python crean cuando se definen por primera vez, es decir, cuando se le asigna un valor por primera vez. Para asignar un valor en una variable se utiliza el operador de igualdad (=). A la izquierda se escribe el nombre de la variable y a la derecha el valor que se quiere dar a la variable.

En el ejemplo siguiente se almacena el número decimal 2.5 en una variable de nombre x

```
>>> x = 2.5
```

La variable se crea escribiendo siempre a la izquierda de la igualdad. Si se escribe al revés, Python genera un nombre de error.

Para que el IOLE muestre el valor de la variable, basta con escribir su nombre

```
>>> x = 2.5
>>> x
2.5
```



Si una variable nombra a sí misma recursivamente, o se intenta usar nombre genera un mensaje de error:

```
>>> x = -10
```

```
>>> y
```

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

y

NameError: name 'y' is not defined

Una variable que almacena un string, texto o estructuras más complejas. Si se van a almacenar textos, el texto se debe de escribir entre comillas simples (' ') o doble (" "), se recomienda las

```
>>> nombre = "Pepe congo"
```

```
>>> nombre
```

'Pepe congo'

Si no se escriben comillas, Python sopara los errores haciendo referencia a otra variable (que, si no está definida, genera un mensaje de error):

```
>>> nombre = Pepe
```

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

nombre = Pepe

NameError: name 'Pepe' is not defined

```
>>> nombre = Pepe congo
```

SyntaxError: invalid syntax



→ Bonos una variable

La introducción del bando completamente una variable.

>>> nombre = "Pepito cargo"
>>> nombre
'Pepito cargo'
>>> del nombre
>>> nombre
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
 nombre
NameError: name 'nombre' is not defined

→ Nombre de variable

Aunque no es obligado se recomienda que el nombre de la variable esté relacionado con la información que se almacena en ella, para que sea más fácil entender el programa. Si el programa es liviano o en inglés, si esta escribiendo en programación, esto no aparece muy importante, pero si se consolida en programación.

El nombre de una variable debe comenzar por una letra o por un guion bajo (-) y puede seguir con más letras, números o guiones bajos.

>>> _x = 3.8 >>> x1 = 100
>>> -x >>> X
3.8 100

>>> fecha_de_nacimiento = "14 de abril del 2000"
>>> fecha_de_nacimiento
'14 de abril del 2000'



Los nombres de variables no pueden contener espacios en blanco:

>>> fecha de nacimiento - "14 de abril del 2000"
SyntaxError: invalid syntax

Los nombres de variables pueden contener cualquier carácter alfabético (los del alfabeto inglés, pero también ñ, e o vocales con tilde), aunque se recomienda utilizar únicamente las características del alfabeto inglés

>>> año = 1997

Nota: En Python 2 los nombres de las variables no podrán contener caracteres no ingleses

Los nombres de las variables pueden contener mayúsculas, pero lo que sucede es que Python distingue entre mayúsculas y minúsculas (en inglés se dice que Python es case-sensitive)

```
>>> nombre = "Pepito congo"
>>> Nombre = "Numa nigento"
>>> nombre = "Fulanito nena"
>>> nombre
'Pepito congo'
>>> Nombre
'Numa nigento'
>>> nombre
'Fulanito nena'
```



Cuando el nombre de una variable contiene varias palabras, se acostumbra separarlas con guiones bajos para facilitar la legibilidad, aunque también se utiliza la notación ~~como Case~~, en la que las palabras no se separan pero empiezan con mayúsculas (salvo la primera palabra):

>>> fecha_de_nacimiento = "14 de abril de 2000"
>>> fechaDeNacimiento = "14 de abril de 2000"

Los palabras reservadas del lenguaje (las que el IDE escribe en negrita) están prohibidas como nombres de variables.

>>> lambda = 3
SyntaxError: invalid syntax

Los nombres de las funciones integradas si se pueden utilizar como nombres de variables, pero más vale no hacerlo porque a continuación ya no se puede utilizar la función como tal:

>>> print = 3
>>> print("Hola")
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
print("Hola")
TypeError: 'int' object is not callable

Borrando con ~~del~~ la variable con nombre de función, se recupera la función

>>> del print
>>> print("Hola")
Hola



- Tipos de variables

En el apartado anterior hay distinciones de algunos de los tipos de variables que hay en Python: números decimales, números enteros y cadenas (strings o más bien listas).

Aunque se observan algunas similitudes, para Python no es lo mismo un número entero, un número decimal o una cadena ya que, por ejemplo, los números se pueden multiplicar por dos cadenas (conocémoslo, una cadena si se puede multiplicar por un número).

Por tanto, estas tres diferencias de variables no son equivalentes.

```
>>> flecha = 1991
>>> flecha = 1991.0
>>> flecha = "1991"
>>> flecha = [27, "Octubre", 1991]
```

En el primer caso la variable flecha está almacenando un número entero en el segundo flecha está almacenando un número decimal y en el tercero flecha está almacenando una cadena de cuatro letras. En el cuarto, la flecha está almacenando una lista (un tipo de variable que puede contener varias otras variables ordenadas).



- Mostrar el valor de una variable en IDLE

Para que el IDLE muestre el valor de una variable, basta con escribir su nombre.

>>> a = 2

>>> a

2

También se pueden conocer el valor de la variable, o la vez escritas juntas, entre comillas (IDLE las mostraría en parentesis, porque Python las considera mediante un contenedor ordenado llamado tupla), como muestra el siguiente ejemplo:

>>> a = b = 2

>>> c = "pepe"

>>> a

2

>>> c, b

('pepe', 2)

- Utilizar o modificar una variable ya definida

Una vez ya definido una variable, se puede utilizarla para hacer cálculos o para hacer reasignaciones o para definir nuevas variables:

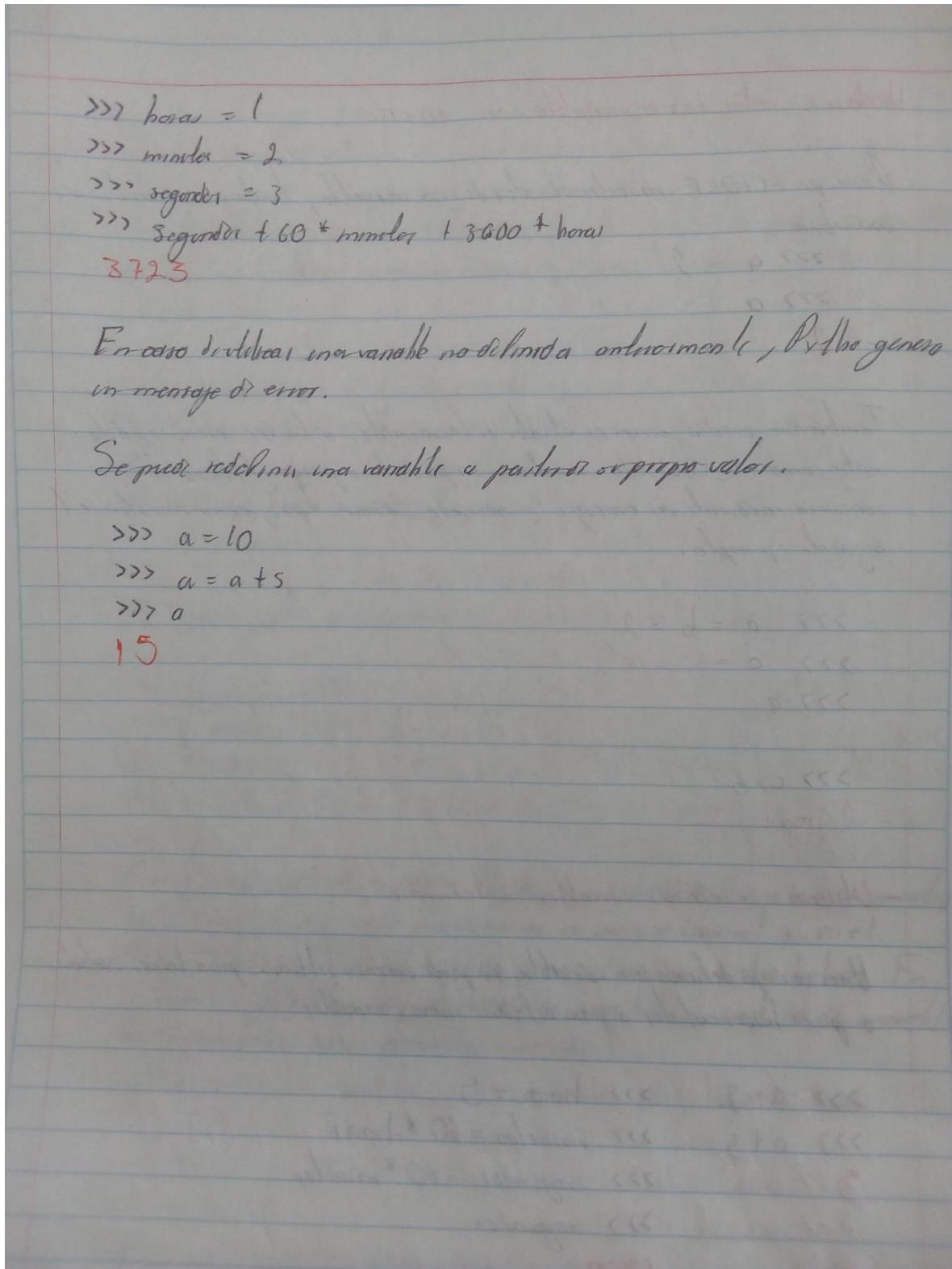
>>> a = 2 >>> horas = 5

>>> a + 3 >>> minutos = 60 * horas

5 >>> segundos = 60 * minutos

>>> segundos

1800





Booleanos

Típo booleanos

Los ejemplos de cosa hacen mucha el resultado de escrita, las operaciones en el prompt de IPLE

Típo booleanos: true y False

Una variable booleana es una variable que puede tomar dos posibles valores: true (verdadero) o False (falso).

En Python cualquier variable (en general, cualquier objeto) puede considerarse como una variable booleana. En general los elementos vacíos o nulos se consideran False y el resto se considera True.

Para considerar si un elemento se considera True o False, se puede convertir su valor booleano mediante la función bool().

>>> bool(0)

False

>>> bool(0.0)

False

>>> bool("")

False

>>> bool(None)

False

>>> bool(C)

False

>>> bool([])

False

>>> bool({})

False

>>> bool(25)

True

>>> bool(-9.5)

True

>>> bool("abc")

True

>>> bool((1, 2, 3))

True

>>> bool([27, "octubre", 1997])

True

>>> bool({27, "octubre", 1997})

True



Operadores lógicos

Los operadores lógicos son unas operaciones que trabajan con valores booleanos.

and "y" lógico. Este resultado da como resultado True si y solo si sus dos operandos son True:

>>> True and True

True

>>> True and False

False

>>> False and True

False

>>> False and False

False

or "o" lógico. Este operador da como resultado True si algún operador es True:

>>> True or True

True

>>> True or False

True

>>> False or True

True

>>> False or False

False

Nota: En el lenguaje cotidiano, el "o" se utiliza a menudo en situaciones en las que se puede optar por una de las alternativas. Por ejemplo, en un menú de restaurante se pide



lograste "postre a café", pero no los dos carros (salvo que se apague aparte, claro). En lógica, este tipo de "o" se denomina "o exclusivo" (XOR).

not negación. Este operador da como resultado True si y solo si su argumento es False

>>> not True

False

>>> not False

True

- Expresiones compuestas

Si no se está acostumbrado a evaluar expresiones lógicas compuestas se recomienda utilizar parentesis para asegurar el orden de las operaciones.

Al comprender expresiones mas complejas, hay que tener en cuenta que Python evalúa primero los not, luego los and y por último los or, como puede comprobarse en los ejemplos segun-

El operando not se evalua ante que el operador and

>>> not True and False

False

>>> (not True) and False

False

>>> not (True and False)

True



El operador `not` se evalúa antes que el operador `or`:

```
>>> not False or True
True
>>>(not False) or True
True
>>> not (False or True)
False
```

El operador `and` se evalúa antes que el operador `or`:

```
>>> False and True or True
True
>>>(False and True) or True
True
>>> False and (True or True)
False
>>> True or True and False
True
>>>(True or True) and False
False
>>> True or (True and False)
True
```

Si en las expresiones lógicas se utilizan valores distintos de `True` o `False`, Python utiliza esos valores en vez de `True` o `False`.

```
>>> 3 or 4
3
```



Si se juntan muchos valores booleanos se puede convertir en un solo valor booleano:

>>> 3 or 4

3

>>> bool(3 or 4) # Verdadero porque 3 es diferente de 0
True

- Comparaciones.

Las comparaciones también son como mezclar valores booleanos

> Mayor que; < Menor que:

>>> 3 > 2

True

>>> 3 < 2

False

>= Mayor o igual que; <= Menor o igual que:

>>> 2 >= 1 + 1

True

>>> 4 - 2 <= 1

False

== Igual que; != Distinto de;

>>> 2 == 1 + 1

True

>>> 6 / 2 != 3

False



Secuencias: Tupla, Rangos, Lista.

Tuplas:

¿Qué son las Tuplas?

En Python, una Tupla es un ~~conjunto~~ ordenado e inmutable de elementos del mismo o diferente tipo.

Las tuplas se representan escribiendo los elementos entre paréntesis y separados por comas.

```
>>> (1, "a", 3.14)  
(1, "a", 3.14)
```

En realidad no es necesario escribir los paréntesis para indicar que se trata de una tupla, basta con escribir las comas, pero Python escribe siempre los paréntesis:

```
>>> 1, "a", 3.14  
(1, "a", 3.14)
```

La función `len()` devuelve el número de elementos de una tupla:

```
>>> len((1, "a", 3.14))  
3
```

Una tupla puede ~~contener~~ no contener ningún elemento, es decir, ser una tupla vacía:

```
>>> ()  
()  
>>> len(())  
0
```



Una tupla puede tener un único elemento, pero para que Python entienda que los elementos referentes a una tupla es necesario escribir al menos una coma.

El ejemplo siguiente muestra la diferencia entre escribir o no una coma. En el primer caso Python interpreta lo expresado como un número y el segundo como una tupla o un único elemento:

```
>>> (3)
3
>>> (3, )
(3,)
```

Python escribe una coma al final en las tuplas de un solo elemento para indicar que se trata de una tupla, pero esa coma no indica que hay un elemento después.

```
>>> (3, )
(3, )
>>> len((3, ))
1
```



Lisitas de números enteros: el tipo range()

En Python 3, range es un tipo de dato. El tipo range es una lista inmutable de números enteros en sucesión aritmética.

Lisita

Las lisitas (list) son un tipo de dato muy flexible que se conoce en la lección lisitas. Como para ver los valores de un rango si se necesita convertirlo a una lista, se conoce que lo define una lista.

Una lista es un conjunto ordenado de elementos del mismo o diferente tipo, cuyo contenido puede modificarse.

Se presenta escribiendo los elementos entre corchetes y separados por comas.

Las variables de tipo lista hacen referencia a la lista completa.

```
>>> lista = [1, "abcde", 45.5, -32]
>>> lista
[1, "abcde", 45.5, -32]
```

Una lista que no contiene ningún elemento se denomina lista vacía:

```
>>> lista = []
>>> lista
[]
```



El tipo range

El tipo **range** es una lista inmutable de números enteros en sucesión aritmética.

- inmutable significa que, a diferencia de las listas, los range no se pueden modificar.
- Una sucesión aritmética es una sucesión en la que la diferencia entre dos términos consecutivos es siempre la misma.

Un **range** se crea llamando al tipo de dato con uno, dos o tres argumentos numéricos, como si fuera una función.

Nota: En Python 2, `range()` se consideraba una función, pero en Python 3 no se considera una función, sino un tipo de dato, aunque se utilice como si fuera una función.

El tipo **range()** con un único argumento se escribe con **range(n)**, y crea una lista inmutable de n números enteros consecutivos que empieza en 0 y acaba en $n - 1$.

Para ver los valores del **range()**, es necesario convertirlos a lista mediante la función **list()**.

```
>>> x = range(10)
>>> x
range(0, 10)
>>> list(x)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(7)
range(0, 7)
>>> list(range(7))
[0, 1, 2, 3, 4, 5, 6]
```



Si n no es positivo, se crean range vacío

>>> list(range(-2))

[]

>>> list(range(0))

[]

El tipo range con dos argumentos se escribe $range(m, n)$ y crea una lista inmutable de enteros consecutivos que empieza en m y acaba en $n - 1$.

>>> list(range(5, 10))

[5, 6, 7, 8, 9]

>>> list(range(-5, 1))

[-5, -4, -3, -2, -1, 0]

Si n es menor o igual que m , se crea un range vacío

>>> list(range(5, 1))

[]

>>> list(range(3, 3))

[]

El tipo range con tres argumentos se escribe $range(m, n, p)$ y crea una lista inmutable de enteros que empieza con m y acaba justo antes de empieza a saltar o iguala a n , saltando los valores $> p$. Si p es negativo, los valores van disminuyendo y se pone.

>>> list(range(5, 21, 3))

[5, 8, 11, 14, 17, 20]

>>> list(range(10, 0, -2))

[10, 8, 6, 4, 2]



El valor de p no puede ser cero:

>>> range(4, 18, 0)

Traceback (most recent call last): /span>

File "<pyshell#0>", line 1, in <module>

range(4, 18, 0)

ValueError: range() arg 3 must not be zero

Si p es positivo y n menor o igual que m, o si p es negativo y n mayor o igual que m, se crea un range vacío.

>>> list(range(25, 20, 2))

[]

>>> list(range(20, 25, -2))

[]

En los range(m, n, p), se pueden combinar p range distintos que generan el mismo resultado. Por ejemplo:

>>> list(range(10, 20, 3))

[10, 13, 16, 19]

>>> list(range(10, 21, 3))

[10, 13, 16, 19]

>>> list(range(10, 22, 3))

[10, 13, 16, 19]

En resumen los tres argumentos del tipo range(m, n, p) son:

- m: el valor inicial.

- n: el valor final (que no se alcanza ni se supera).

- p: el paso (la cantidad que se avanza cada vez).



Si se escriben dos argumentos, Python los asigna a los valores 1. Es decir `range(1, n)` es lo mismo que `range(0, n, 1)`.

Si se crean solo un argumento, Python lo asigna como el valor 0 y a `n` el valor 1. Es decir `range(n)` es lo mismo que `range(0, n, 1)`.

El tipo `range()` solo admite argumentos enteros. Si se utilizan argumentos decimales, se producen errores.

Concatenar `range()`

No se puede concatenar tipos `range()`, ya que el resultado de la concatenación puede no ser de ningún tipo `range()`.

`>>> range(3) + range(5)`

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>
`range(3) + range(5)`

Type Error: unsupported operand type() for +: 'range' and 'range'

Pero si se pone concatenar tipos `range()` previamente convertidos en lista. El resultado es lógicamente una lista, que no se puede convertir a tipo `range()`.

`>>> list(range(3)) + list(range(5))`

`[0, 1, 2, 0, 1, 2, 3, 4]`



No se producen concatenaciones (uso de `+`), ni aunque el resultado sea una lista se suman elementos en sucesiones aritméticas.

>>> `range(1,3) + range(3,5)`

Traceback (most recent call last):

File "<ipython>", line 1, in <module>
`range(3) + range(5)`

TypeError: unsupported operand type(s) for +: 'range' and 'range'

>>> `list(range(1,3)) + list(range(3,5))`

`[1, 2, 3, 4]`

La función `len()`

La función `len()` devolver la longitud de una cadena de caracteres o el número de elementos de una lista. El argumento de la función `len()` es la lista o cadena que queremos "medir".

>>> `len("mensaje secreto")`

15

>>> `len(["a", "b", "c"])`

3

>>> `len(range(1, 100, 7))`

15

El valor devuelto por el método la función `len()` se puede usar como parámetro de `range`.

>>> `list(range(len("mensaje secreto"))))`

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]`

>>> `list(range(len(["a", "b", "c"])))`

`[0, 1, 2]`

>>> `list(range(len(1, 100, 7))))`

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]`



Lisitas

Las lisas son una estructura de datos muy útiles. Python permite manipular lisas de muchas maneras.

Lisitas

Las lisas son conjuntos ordenados de elementos (números, cadenas, lista, etc). Las lisas se crean con paréntesis [] y los elementos se separan por comas.

Las lisas pueden contener elementos del mismo tipo

```
>>> primos = [2, 3, 5, 7, 11, 13]
```

```
>>> diasLaborables = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"]
```

O pueden contener elementos de tipos distintos:

```
>>> fecha = [{"Lunes", 27, "Octubre", 1997}]
```

O pueden combinar lisas:

```
>>> peliculas = [{"Sendas de Gloria", 1957}, {"Hannah y sus hermanas", 1986}]
```

Las lisas pueden tener muchos tipos si anidárnoslos:

```
>>> direcciones = [{"Stanley Kubrick", {"Sendas de Gloria", 1957}}, {"Woody Allen", {"Hannah y sus hermanas", 1986}}]
```

Todos los tipos de lista hacen referencia a la lista completa.

```
>>> lisa = [1, "a", 45]
```

```
>>> lisa
[1, "a", 45]
```



Una lista que no cambie ningún elemento se denomina lista rígida:

>>> lista = []

>>> lista

[]

Al declarar una lista se pueden hacer referencias a otras variables

>>> nombre = "Pepe"

>>> edad = 25

>>> lista = [nombre, edad]

>>> lista

['Pepe', 25]

Como siempre, hay que tener cuidado al modificar una variable se alterará para todos las demás variables, porque esto puede afectar a las otras variables.

* Si se hace de objetos inmutables, solo modificar las variables no resulta alterada, como se muestra en el ejemplo:

>>> nombre = "Pepe"

>>> edad = 25

>>> lista = [nombre, edad]

>>> lista

['Pepe', 25]

>>> nombre = "Juan"

>>> lista

['Pepe', 25]



Por ejemplo, si se trata de objetos mutables, y al modificarlo la variable se modifica el objeto, el valor de la variable no resulta afectada;

```
>>> nombres = [ "Ana", "Bernardo" ]
>>> edades = [ 22, 21 ]
>>> lista = [nombres, edades]
>>> lista
[[ "Ana", "Bernardo" ], [ 22, 21 ]]
>>> nombres += [ "Christina" ]
>>> lista
[ [ 'Ana', 'Bernardo', 'Christina' ], [ 22, 21 ] ]
```

Una lista puede contener otras (que a su vez pueden contener otras, que a su vez, etc.):

```
>>> persona1 = [ "Ana", 25 ]
>>> persona2 = [ "Bentito", 23 ]
>>> lista = [ persona1, persona2 ]
>>> lista
[ [ 'Ana', 25 ], [ 'Bentito', 23 ] ]
```

Se pueden acceder a cualquier elemento de una lista escribiendo el nombre de la lista y entre corchetes el número de orden en la lista. El primer elemento de la lista es un número 0.

```
>>> lista = [ 10, 20, 30, 40 ]
>>> lista[2]
30
>>> lista[0]
10
```



- Concatenar listas.

Las listas se pueden concatenar con el símbolo de la suma (+):

>>> vocales = ["E", "I", "O"]

>>> vocales

['E', 'I', 'O']

>>> vocales = vocales + ["U"]

>>> vocales

['E', 'I', 'O', 'U']

>>> vocales = ["A"] + vocales

>>> vocales

['A', 'E', 'I', 'O', 'U']

El operador suma (+) necesita que los dos operadores sean listas:

>>> vocales = ["E", "I", "O"]

>>> vocales = vocales + "Y"

Traceback (most recent call last):

File "<ipython#2>", line 1, in <module>

vocales = vocales + "Y"

TypeError: can only concatenate list (not "str") to list

También se puede utilizar, los operadores += para añadir un elemento a la lista

>>> vocales = ["A"]

>>> vocales += ["E"]

>>> vocales

['A', 'E']

En estos ejemplos, los operadores + y += de el mismo resultado, no son equivalentes.



Manipular elementos individuales en una lista

Cada elemento se identifica por su posición en la lista, comenzando en cero lo que corresponde a contar por 0

>>> Fecha = [27, "Octubre", 1997]
>>> Fecha = [0]
27
>>> Fecha = [1]
Octubre
>>> Fecha = [2]
1997

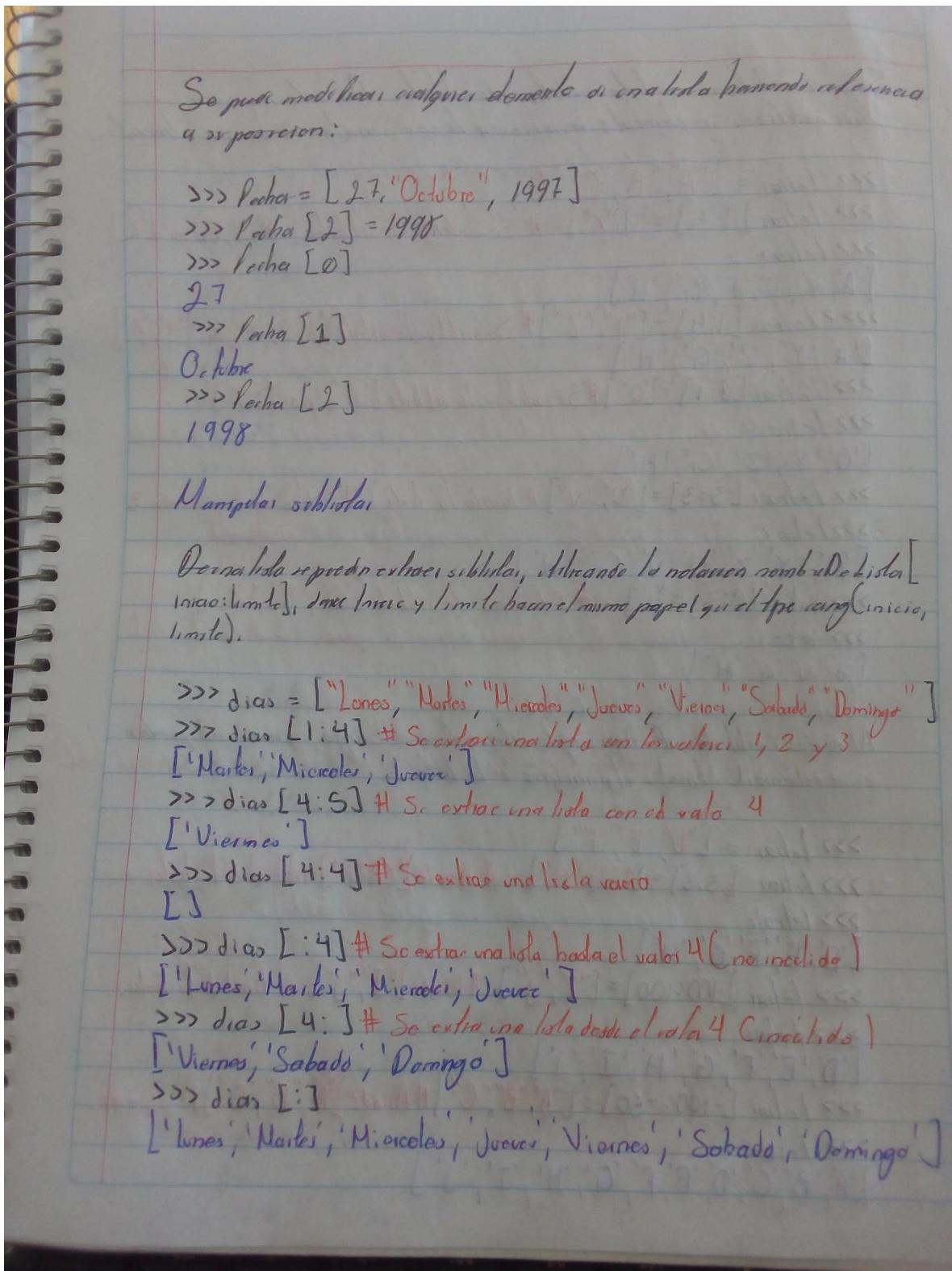
No se pierde hacer referencia a elementos usando la lista:

>>> fecha = [27, "Octubre", 1997]
>>> fecha[3]

Traceback (most recent call last):
[1] <ipython-input-3>, line 1, in <module>
 Fecha[3])
IndexError: list index out of range

Se pueden extraer números negativos (el último elemento tiene el índice -1 y los valores anteriores tienen valores descendentes):

>>> Fecha = [27, "Octubre", 1997]
>>> Fecha = [-1]
1997
>>> Fecha = [-2]
Octubre
>>> Fecha = [-3]
27





Se puede modificar una lista mediante sublistas. De esta manera, se puede modificar un elemento o varios a la vez, sin tener que borrar el elemento.

```

>>> letras = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
>>> letras[1:4] = ['X'] # Se sustituye la sublista ['B', 'C', 'D'] por ['X']
>>> letras
['A', 'X', 'E', 'F', 'G', 'H']
>>> letras[1:4] = ['Y', 'Z'] # Se sustituye la sublista ['E', 'F'] por ['Y', 'Z']
['A', 'Y', 'Z', 'G', 'H']
>>> letras[0:1] = ['Q'] # Se sustituye la sublista ['A'] por ['Q']
>>> letras
['Q', 'Y', 'Z', 'G', 'H']
>>> letras[3:3] = ['U', 'V'] # Inserta la lista ['U', 'V'] en la posición 3
>>> letras
['Q', 'Y', 'Z', 'U', 'V', 'G', 'H']
>>> letras[0:3] = [] # Elimina la sublista ['Q', 'Y', 'Z']
>>> letras
['U', 'V', 'G', 'H']

Al definir sublistas, Python acepta valores fuera del rango, que se interpretan como extremos (al final o al principio de la lista).

```

```

>>> letras = ['D', 'E', 'F']
>>> letras[3:3] = ['G', 'H'] # Agrega ['G', 'H'] al final de la lista
>>> letras
['D', 'E', 'F', 'G', 'H']
>>> letras[100:100] = ['I', 'J'] # Agrega ['I', 'J'] al final de la lista
>>> letras
['D', 'E', 'F', 'G', 'H', 'I', 'J']
>>> letras[-100:-50] = ['A', 'B', 'C'] # Agrega ['A', 'B', 'C'] al principio
>>> letras
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']

```



La palabra reservada del

La palabra reservada del permite eliminar un elemento o varios elementos, a lo largo de una lista, e incluso la misma lista.

```
>>> letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
>>> del letras[4] # Elimina la sublista [F]
>>> letras
['A', 'B', 'C', 'D', 'F', 'G', 'H']
>>> del letras[1:4] # Elimina sublista [B, C, D]
>>> letras
['A', 'F', 'G', 'H']
>>> del letras # Elimina completamente la lista
>>> letras
```

Traceback (most recent call last):
 File "<pyshell#1>", line 1, in ?
 letras

NameError: name 'letras' is not defined

Si se intenta borrar un elemento que no existe, se produce un error.

```
>>> letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
>>> del letras[10]
Traceback (most recent call last):  

  File "<pyshell#1>", line 1, in <module>
    del letras[10]
IndexError: list assignment index out of range
```

Aunque si se hace referencia a sublistas, Python sí acepta valores fuera de rango, pero lógicamente no se modifica la lista.

```
>>> letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
>>> del letras[100:200] # No elimina nada
>>> letras
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
```



Copia una lista

Con variables de tipo entero, decimal o cadena, es fácil tener una copia de una variable para conservar el valor que tiene la variable original si ha pasado:

```
>>> a = 5
>>> b = a # Hacemos una copia del valor q
>>> a, b
(5, 5)
>>> a = 4 # de manera q aunque cambiemos el valor de a ...
>>> a, b # ... b conserva el valor anterior de a en caso de necesitarlo
(4, 5)
```

Pero si hacemos esto con listas, no podemos llevar una sorpresa:

```
>>> lista1 = ["A", "B", "C"]
>>> lista2 = lista1 # La idea es hacer una copia de la lista lista1
>>> lista1, lista2
([['A', 'B', 'C']], ['A', 'B', 'C'])
>>> del lista1[1] # Eliminamos el elemento ['B'] de la lista 1
>>> lista1, lista2 # ... Pero también descubrimos q también ha desaparecido lista2
([['A', 'C']], ['A', 'C'])
```

El motivo de este comportamiento, es q q los enteros, decimales y cadenas son objetos mutables y las listas son objetos inmutables.



Si queremos copiar una lista, de manera que conservemos su valor original, modifiquemos la lista original y debemos utilizar la notación de `listas`,

`>>> lista1 = ["A", "B", "C"]`

`>>> lista2 = lista1[:] # Hacemos una copia de la lista lista1`

`>>> lista1, lista2`

`(['A', 'B', 'C'], ['A', 'B', 'C'])`

`>>> del lista1[1] # Eliminamos el elemento ['B'] de la lista1`

`>>> lista1, lista2 # ... y en esta caso lista2 sigue conservando el valor original`
`(['A', 'C'], ['A', 'B', 'C'])`

Estructura de control: Sentencias condicionales (if, elif, else).

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named `EdadIf_elif_else.py` containing the following code:

```

1 # -*- coding: utf-8 -*-
2 """
3     Created on Wed Jun 17 19:42:05 2020
4
5     @author: Miguel Angel
6
7
8     Edad = int(input("Que edad tiene: "))
9
10 if Edad > 18:
11     print("Usted tiene la edad suficiente.")
12 elif Edad >= 18 and Edad <= 18:
13     print("Usted tiene la Edad Necesaria")
14 else:
15     print("Usted no corresponde con la edad suficiente")

```

The right side of the interface shows a plot of a sine wave from 0 to 20. Below the plot, the IPython terminal window shows the execution of the script and its output:

```

Que edad tiene: 17
Usted no corresponde con la edad suficiente
In [1]: runfile('C:/Users/Miguel Angel/EdadIf_elif_else.py', wdir='C:/Users/Miguel Angel')
Que edad tiene: 18
Usted tiene la Edad Necesaria
In [2]: runfile('C:/Users/Miguel Angel/EdadIf_elif_else.py', wdir='C:/Users/Miguel Angel')
Que edad tiene: 19
Usted tiene la edad suficiente.
In [3]:

```

Imagen 1. Estructura (if, elif, else) en Python.

Iteraciones (bucle for)

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named `BucleFor.py` containing the following code:

```

1 # -*- coding: utf-8 -*-
2 """
3     Created on Wed Jun 17 21:31:06 2020
4
5     @author: Miguel Angel
6
7
8
9     for x in [0,1,2,3,4,5,6,7,8,9,10]:
10         print(x)
11
12

```

The right side of the interface shows a plot of a sine wave from 0 to 20. Below the plot, the IPython terminal window shows the execution of the script and its output:

```

Escrba un numero positivo: 5
El numero que usted escribio es un numero positivo
In [9]: runfile('C:/Users/Miguel Angel/BucleFor.py', wdir='C:/Users/Miguel Angel')
0
1
2
3
4
5
6
7
8
9
10
In [10]:

```

Imagen 2.Estructura For en Python.

Iteraciones (bucle while)

The screenshot shows the Spyder Python 3.7 IDE interface. On the left, the code editor displays a script named `Sin titulo2.py` containing a `while` loop that checks if a number is positive or negative. The code is as follows:

```
1  # -*- coding: utf-8 -*-
2  """
3      Created on Wed Jun 17 22:03:01 2020
4
5      @author: Miguel Angel
6
7
8
9      numero = int(input("Escriba un numero positivo: "))
10     * while numero < 0:
11         print("Ha escrito un numero negativo\nIntentalo de nuevo")
12         numero = int(input("Escriba un numero positivo: "))
13         print("El numero que usted escribio es un numero positivo")
14
15
```

To the right of the code editor is a plot window showing a sine wave from 0 to 20 on the x-axis and -1.00 to 1.00 on the y-axis. Below the plot is a terminal window showing the execution of the script and user interaction. The terminal output is:

```
In [8]: runfile('C:/Users/Miguel Angel/Sin titulo2.py', wdir='C:/Users/Miguel Angel')
Escriba un numero positivo8
Ha escrito un numero negativo
Intentalo de nuevo
Escriba un numero positivo: -5
Ha escrito un numero negativo
Intentalo de nuevo
Escriba un numero positivo: 5
El numero que usted escribio es un numero positivo
In [9]:
```

The bottom status bar indicates the environment is conda: base (Python 3.7.6), line 11, col 61, and the time is 10:16 p.m. on June 17, 2020.

Imagen 3. Estructura While en Python.

Conclusión

En este apartado tuvimos que ver ciertos puntos que debemos conocer, ya que, si tenemos que programar en Python, tenemos que tener en cuenta cuales o como es que funcione el lenguaje Python. Ya que como nosotros estamos más especializados en el lenguaje Java, que en la de Python. Entonces esta vez tuvimos que saber cómo programar, ya que en Python es conocido por su forma de trabajar, ya que está relacionado con la programación orientado a objetos, que es lo que mas se relaciona con la vida cotidiana.

Fue necesario investigar los puntos más importantes y otros puntos que ya sabíamos, pero de una forma también relacionarnos más, vimos que no se diferencia mucho en Java, pero si se trabaja de una forma muy diferente a Java. Una de las cosas que vimos en Python fue la forma en como se trabaja con los números enteros y de tal forma como podemos realizar operaciones, ya sea las operaciones básicas, problemas de álgebra, ecuaciones. Etc. Pero también cuales son los posibles resultados que podemos tener si no programamos de forma correcta y nos puede salir un error de programación o si no de sintaxis. De ahí vimos cómo se trabaja con las cadenas de texto, las cuales son una parte esencial al momento de programación, ya que podemos mostrar información o un mensaje, y la diferencia que tiene en Java, ya que en Python se ocupa las comillas simples y dobles, y de una forma como interactuar con las dos, de tal forma también vimos los posibles errores que podríamos tener si no tomamos en cuenta como programar las cadenas de texto. Otra Parte importante fue conocer las variables en Python, pero antes que nada saber que es una variable y cómo podemos trabajar con dichas variables, porque si bien sabemos la variable se ocupa en las ecuaciones, álgebra líneas, etc. Y de tal forma que se representa con una letra “X”. tanto en Java y en Python se ocupa una variable llamada Booleano donde esta variable se ocupa para situaciones en donde cierta forma si una respuesta es negativa (Falso) el programa no termina hasta que la respuesta sea positiva (Verdadero), entonces el programa termina, pero no siempre es así porque hay caso que es de una (Falso o Verdadero).



Bibliografía

Python. (s.f.). Recuperado el 24 de Junio de 2020, de Python:

<https://www.mclibre.org/consultar/python/lecciones/python-operaciones-matematicas.html>

Python. (s.f.). Recuperado el 24 de Junio de 2020, de Cadenas de texto:

<https://www.mclibre.org/consultar/python/lecciones/python-cadenas.html>

Python. (s.f.). Recuperado el 24 de Junio de 2020, de Variables:

<https://www.mclibre.org/consultar/python/lecciones/python-variables.html>

Python. (s.f.). Recuperado el 24 de Junio de 2020, de Tipos de Booleanos:

<https://www.mclibre.org/consultar/python/lecciones/python-booleanos.html>

Python. (s.f.). Recuperado el 24 de Junio de 2020, de Listas:

<https://www.mclibre.org/consultar/python/lecciones/python-range.html>