Clase_1_Variables

May 24, 2022

1 Curso de Python

1.1 por Carlos Feinstein, Cursada 2020

El lenguaje python cambia seguido así que lo que se indica en este apunte tiene una duración en el tiempo y por lo tanto fecha de vencimiento. Este apunte es para el uso de las versiones Python 3, no funciona en las versiones anteriores. Ya que en el Python no hay compatibilidad con versiones previas (backwards compatibility)

Este curso se basa en el uso de python sobre un Notebook del entorno de trabajo Jupyter (aunque veremos que funciona bien en el entorno propietario de Google llamado Colaborative.

Cómo cualquier otro lenguaje nuevo que quiere aprender las primeras cosas que hay que entender son las variables y como se realizan las asignaciones.

2 Variables en Python y manejo de datos

2.1 Variables simples

Las variables de los sentencias python tiene una diversidad muy grande, están las básicas de del python original, pero que se complementan con otras que son agregadas por distintas librerías (numpy, pandas, astropy, etc..) que son de uso muy común en el análisis de datos. Dado esta diversidad para evitar confuciones existen órdenes para preguntar a una variable de que tipo es (por ejemplo: type()). Veamos cada una de ellas, empezando por las básicas que son muy parecidas a las de los otros lenguajes. Pero hay siempre tener presente que Python es un leguaje orientado a objetos, así que las variables son objetos. Y que los objetos pueden ser creados por el ususario, es decir, uno puede crear variables con las propiedades que necesita.

Tipos de datos básicos:

- Enteros (Integers)
- Flotantes (Floats)
- Textos (Strings)
- Lógicos (Boolean)
- Números complejos

2.1.1 Comentarios

En python hay como otro lenguaje la posibilidad de poner comentarios que no afectan la corrida del programa pero muy útiles en la documentación de las intenciones del programador. Veamos su uso:

A partir de un "#" todo lo que sigue es un comentario que se acaba con el renglón

En cambio 3" producen comentarios que necesitan indicar su final y pueden ocupar varios renglones.

""" Este en cambio es un comentario que ocupa muchas líneas """

2.1.2 Números

[3]:
```
uno = 1 # Así sería entero (ojo no es 1. -> no tiene el ".") <-- esto fue un␣
 ↪comentario!!!
print(uno)

# La orden type me sirve para preguntar que tipo de variable es. En este caso en␣
 ↪particular
# es un entero.
# preste atención a la orden print(), en al cual el texto pueden ponerse con ¨␣
 ↪(o también '), para que
# salga como tal cual y que las distintas variables se separan con ",".

print('Esa variable es del tipo', type(uno))
```

```
1
Esa variable es del tipo <class 'int'>
```

[4]:
```
uno_f =1.
print(uno,uno_f)
# El 1 flotante es 1.0 (con el . y el 0, mientras que en el entero no)

print('Esa variable es del tipo', type(uno_f))
```

```
1 1.0
Esa variable es del tipo <class 'float'>
```

Ojo hay nombres que son prohibidos, no se permite su uso como variables o funciones, ya que identifican órdenes del lenguaje Python Este es el listado de esos nombres:

and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield True, False, None <– Estas últimas tres son las únicas en mayúsculas

En los nombres de las variables no pueden estar estos símbolos: !, @, #, $, %

El nombre de una variable no puede empezar con un número, pero pueden tenerlo dentro del nombre, se puede hacer variables con nombres largos, y a veces es cómodo separar las palabras con el símbolo (_) que se convierte parte del nombre

Que_feo_numero = 171717171.71717

2.1.3 Sentencias multilínea

Hay que prestar atención a la sangria (identation) y a los símbolos “, (), [] y {}” La sangría en Python significa que la sentencia está relacionada o englobada en la orden anterior. A diferencia de los demás lenguajes de computación su uso indica una acción, no es “inerte” como en la mayoría de los otros lenguajes. Tiene significado funcional y hace cosas.

[6]:
```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9

b = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)

# también iniciar muchas variables en un sólo renglón

a = 1; b = 2; c = 3

x = y = z = "Lo mismo en las tres variables."

print(x,y)
```

```
Lo mismo en las tres variables. Lo mismo en las tres variables.
```

2.1.4 Strings

Los strings son cadenas de caracteres de texto, y sirven para manejar justamente textos. Se les pueden aplicar muchos comandos y funciones para realizar distintas modificaciones. Es posible cortar o pegar textos, en la sección de operaciones las veremos con detalle.

[7]:
```
curso = 'Curso2020'

print(curso)
```

```
Curso2020
```

[8]:
```
curso[0]
```

[8]:
```
'C'
```

[9]:
```
curso[1]
```

[9]:
```
'u'
```

[10]:
```
curso[0:5]
```

[10]:
```
'Curso'
```

Ya que sería string[inicio:final:paso] <– Esta es una forma de notación muy general en Python
Recordar que no el inicio comienza en "0" y el final en el n-1, si dice "4" el final entonces es el elemento "3" (ojo que igual hay cuatro elementos porque el primero fue el 0.

Si pido un elemento del string correspondiente a un número negativo, estos se empiezan a contar desde el final del número no importa cuantos elementos tenga.

[11]:
```
curso[1:6:2]
```

[11]:
```
'us2'
```

[13]:
```
print('[0] =>', 'Curso'[0])
print('[0:1] =>', 'Curso'[0:1])
print('[0:2] =>', 'Curso'[0:2])
print('[-1] =>', 'Curso'[-1])
```

```
[0] => C
[0:1] => C
[0:2] => Cu
[-1] => o
```

[14]:
```
#### Se pueden incluso usar caracteres UNICODE
```

[1]:
```
strings = "Esto es Python"
char = "C"
multilinea_str = """Este es un string multilínea con más
de una línea de código."""
unicode = 'Yo \u2665 la cursada de Computación'
crudo_str = r"Este está crudo  \n string" # <-- prestar atención al r antes del "
crudo_str2 = "Este está crudo  \n string"

# imprimo el texto directo
print(strings)
print(char)
print("") # <---renglón en blanco
```

```
# en este caso tiene varios renglones al indicarse la asginación en la variable,␣
 ↪pero se unen al asignarse
print(multilinea_str)
print("")

# Se pueden escribir caracteres UNICODE
print(unicode)
print("")

# O hacer que el texto se haga literal (sin ejecutar el "\n" que haria que se␣
 ↪baje de renglón). Esto
#se activa por el "r" antes del texto en la asginación
print(crudo_str)
print(crudo_str2)
```

```
Esto es Python
C

Este es un string multilínea con más
de una línea de código.

Yo  la cursada de Computación

Este está crudo  \n string
Este está crudo
 string
```

2.1.5 Variables Logicas (Booleanas)

Sólo pueden guardar un "Verdadero" (True) o un "Falso" (False). Van con la primera letra en mayúscula.

[16]:
```
al = True
bl = False
print(al,bl)
print(type(al),type(bl))
```

```
True False
<class 'bool'> <class 'bool'>
```

Si bien el resultado es True o False desde el punto de vista numérico hay un "1" en caso de verdadero y un "0" en caso de que sea falso el resultado lógico. Es decir las variables en si tienen guardado un número 1 o un 0 en ellas.

2.1.6 Convirtiendo tipos de Variables

USO las siguientes funciones, por eso llevan ()

- float() –> convierte a flotantes
- int() –> convierte a enteros
- str() –> convierte a strings

[17]:
```
a = True
print(float(a))
print(int(a))
print(int(13.97))
print(str(a))
```

```
1.0
1
13
True
```

2.2 Operaciones con las variables

2.2.1 Operaciones matemáticas Simples

- Son iguales al FORTRAN
- Asignan los resultados de la misma manera a una tercer variable
- Hay operaciones no númericas
- Están las muy básicas y se usan exactamente igual que Fortran, salvo una: +,-,*,/,**
- Y otras no convencionales, por ejemplo: //,%
- Existen asignaciones incrementales (o decrementales) sobre la misma variable (aditivas, multiplicativas)

[2]:
```
a = 2
b = 2
c = a + b
d = a - b
e = a * b
f = a ** b
g = a / b

print(c, 'es', a, '+', b)
print(d, 'es', a, '-', b)
print(e, 'es', a, '*', b)
print(f, 'es', a, '**', b)
print(g, 'es', a, '/', b, '¿Qué notan aquí?')
```

```
4 es 2 + 2
0 es 2 - 2
4 es 2 * 2
```

```
4 es 2 ** 2
1.0 es 2 / 2 ¿Qué notan aquí?
```

2.2.2 Las divisiones entre enteros generan un número real!!!!!

Esto es diferente a Fortran e incluso a las versiones anteriores de Python, incluyendo Python 2

[22]:
```
# ejemplo

124231/1134
```

[22]:
```
109.55114638447972
```

2.2.3 Ojo que se pueden hacer asinaciones múltiples

o sea, que en una sentencia se asignan varios resultados El orden es determinante de a que variable le llega cada dato.

[ ]:
```
x,y = 25.4, 54.67788
print(x,y)

z1,z2= x*24.5,y*4567
print(z1,z2)
```

2.2.4 Hay operaciones no convencionales

[23]:
```
# ojo que existen estas operaciones. Esta es el resultado en
# enteros (floor division)

17//3
```

[23]:
```
5
```

[24]:
```
# y esta es el resto de la división. La función módulo en Fortran
17%3
```

[24]:
```
2
```

[25]:
```
# Operaciones incrementales

i=1

i+=10  #suma 10 a i
print(i)
```

```
i-=2  #resto 2 a i
print(i)

i*=4  #multiplico por 4 a i
print(i)

i/=5  #divido por 5 a i
print(i)
```

```
11
9
36
7.2
```

[26]:
```
# operaciones con streams (textos)

a = 'Hola, estoy en '
b = 'la clase de computación'
x = a + b
print(x)
```

```
Hola, estoy en la clase de computación
```

[28]:
```
print('Recuerdo el valor de la variable "al"->(',al,') de una celda anterior')
print('Si multiplico un Booleano y era verdadero:', al*8)
print('Si multiplico un Booleano y era falso    :' ,bl*8)
```

```
Recuerdo el valor de la variable "al"->( True ) de una celda anterior
Si multiplico un Booleano y era verdadero: 8
Si multiplico un Booleano y era falso    : 0
```

2.2.5 Precision de los cálculos

Es diferente a Fortran, los reales son doble precisión (real*8) y los enteros tienen precisión arbitraria.

Recordar que las variables en Doble Precisión dependen del hardware pero en general los números válidos están entre 10^-308 y 10^308, con 16 digitos significativos

[3]:
```
numero=3242525225
z = numero**8
print(z)
```

```
122198790302863069638607705589529739839448039955553855075819700196838378906 25
```

[4]:
```
a=1
b=100000000000000000000
print(a/b)
```

```
1e-21
```

[5]:
```
# recordar que puedo preguntar de que tipo es la variables y sus datos
print(type(2))
print(type(2.3))

# además puedo cortar (truncar) y redondear los números

print(int(0.8)) # truncando
print(round(0.88766)) # ahora el más cercano pero que sea entero:

# Entendiendo el redondeo, esta busca el par más cercano.

print(round(0.3))
print(round(0.5))
```

```
<class 'int'>
<class 'float'>
0
1
0
0
```

2.2.6 Números complejos

Hay que recordar el concepto de objetos, ahora lo veremos en el caso de números complejos, pero esto se extiende a todos los tipos de variables. En Python la parte imaginaria se indica con la letra j, no con i como es la convención normal que se usa en matemática. Por lo cual un número complejo sería el siguiente:

[36]:
```
a = 1.5 + 0.5j
```

[37]:
```
a**2
```

[37]:
```
(2+1.5j)
```

[38]:
```
a.real
```

[38]:
```
1.5
```

[39]:
```
a.imag
```

[39]:
```
0.5
```

[40]:
```
a.conjugate() # En este caso la función requiere los ()
```

[40]:
```
(1.5-0.5j)
```

[41]:
```
a*a.conjugate()
```

[41]:
```
(2.5+0j)
```

2.2.7 Operaciones entre Booleanos

[42]:
```
5 < 7
```

[42]:
```
True
```

[43]:
```
a = 5
b = 7
```

[44]:
```
b < a
```

[44]:
```
False
```

[45]:
```
c = 2
```

[46]:
```
c < a < b
```

[46]:
```
True
```

Los operados Boleanos (lógica) son “and, or, not” pero hay cosas como “is” (es un “and” literal)

[47]:
```
a < b and b < c
```

[47]:
```
False
```

[48]:
```
res = a < 7
print(res, type(res))
```

```
True <class 'bool'>
```

[49]:
```
print(res)
print(not res)
```

```
True
False
```

[50]:
```
not res is True
```

[50]:
```
False
```

[51]:
```
a = True
print(a)
```

```
True
```

2.2.8 Texto/Strings

[53]:
```
a = "Esto es un    string"
```

[55]:
```
# Pregunto su tamaño

len(a)
```

[55]:
```
20
```

Los strings son objetos (como todo en python).
Recordar que hay métodos (o sea funciones) que están definidas en los objetos:

[56]:
```
a.upper()
```

[56]:
```
'ESTO ES UN    STRING'
```

[57]:
```
a.title()
```

[57]:
```
'Esto Es Un    String'
```

[58]:
```
a.split()
```

[58]:
```
['Esto', 'es', 'un', 'string']
```

[59]:
```
a.split()[1]
```

[59]:
```
'es'
```

[60]:
```
a = "Este es un string.   Con muchas sentencias."
```

[61]:
```
a.split('.')
```

[61]:
```
['Este es un string', '   Con muchas sentencias', '']
```

[62]:
```
a.split('.')[1].strip()
# Defino el caracter para el split .
```

```
#Strip saca los blancos que están demás. El default es el blanco, puede␣
 ↪cambiarse a otro caracter
```

[62]:
```
'Con muchas sentencias'
```

[63]:
```
b = '   <-muchos espacios'
b.strip()
```

[63]:
```
'<-muchos espacios'
```

[64]:
```
a = 'tru'
b = 'la'
print(' '.join((a,b,b)))
print('-'.join((a,b,b)))
print(''.join((a,b,b)))
print(' '.join((a,b,b)).split())
print(' & '.join((a,b,b)) + 'lo')
```

```
tru la la
tru-la-la
trulala
['tru', 'la', 'la']
tru & la & lalo
```

2.2.9 Ingreso de datos por teclado

Se utiliza la función input()

[65]:
```
base = float(input("base: "))
altura = float(input("altura: "))

area=base*altura/2

print("El área es=",area)
```

```
base: 34
altura: 35
El área es= 595.0
```

2.3 Variables mas complejas (en la siguiente clase)

Tipos de datos:

- Listas
- Tuplas
- Diccionarios

[ ]:
```

```

[ ]: