1 Definición de variables.

Objetivo. Explicar el concepto de variable, etiqueta, objetos y como se usan mediante algunos ejemplos.

Funciones de Python: - `print()`, `type()`, `id()`, `chr()`, `ord()`, `del()`

1.1 Variables.

- Son **símbolos** que permiten identificar la información que se almacena en la memoria de la computadora.
- Son **nombres** o **etiquetas** para los objetos que se crean en Python.
- Se crean con ayuda del operador de asignación `=`.
- No se tiene que establecer explícitamente el tipo de dato de la variable, pues esto se realiza de manera dinámica (tipado dinámico).

1.1.1 Ejemplos de variables válidas.

Los nombres de las variables: * pueden contener **letras**, **números** y **guiones bajos**, * deben comenzar con una letra o un guion bajo, * se distingue entre mayúsculas y minúsculas, es decir, `variable` y `Variable` son nombres diferentes.

A continuación se muestran algunos ejemplos.

```
_luis = "Luis Miguel de la Cruz"    # El nombre de la variable es _luis, el conte
LuisXV = "Louis Michel de la Croix"
luigi = 25
luis_b = 0b01110 # Binario
luis_o = 0o12376 # Octal
luis_h = 0x12323 # Hexadecimal

# Sensibilidad a mayúsculas y minúsculas
# los siguientes nombres son diferentes
pi = 3.14
PI = 31416e-4
Pi = 3.141592
```

Podemos ver el contenido de la variable usando la función `print()`:

```
# El contenido de cada variable se imprime en renglones
# diferentes debido a que usamos el argumento sep='\n'
```

```
print(_luis, LuisXV, luigi, luis_b, luis_o, luis_h, pi, PI, Pi, sep='\n')
```

```
Luis Miguel de la Cruz
Louis Michel de la Croix
25
14
5374
74531
3.14
3.1416
3.141592
```

NOTA. Para saber más sobre la función `print()` revisa la sección XXX.

Para saber el tipo de objeto que se creo cuando se definieron las variables anteriores, podemos hacer uso de la función `type()`:

```
print(type(_luis), type(LuisXV), type(luigi),
      type(luis_b), type(luis_o), type(luis_h),
      type(pi), type(PI), type(Pi), sep = '\n')
```

```
<class 'str'>
<class 'str'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'float'>
<class 'float'>
<class 'float'>
```

También es posible usar la función `id()` para conocer el identificador en la memoria de cada objeto como sigue:

```
print(id(_luis), id(LuisXV), id(luigi),
      id(luis_b), id(luis_o), id(luis_h),
      id(pi), id(PI), id(Pi), sep = '\n')
```

```
139672517675408
139672517886160
94157725269672
94157725269320
139672517639248
139672517630576
139672517623920
139672517638992
139672518259056
```

Observa que cada objeto tiene un identificador diferente. Es posible que un objeto tenga más de un nombre, por ejemplo

```
luiggi = _luis
```

La etiqueta o variable `luiggi` hace referencia al mismo objeto que la variable `_luis`, y eso lo podemos comprbar usando la función `id()`:

```
print(id(luiggi))
print(id(_luis))
```

```
139672517675408
139672517675408
```

1.1.2 Ejemplos con Unicode.

Unicode: estándar para la codificación de caracteres, que permite el tratamiento informático, la transmisión y visualización de textos de muchos idiomas y disciplinas técnicas. Unicode intenta tener universalidad, uniformidad y unicidad. Unicode define tres formas de codificación bajo el nombre UTF (Unicode transformation format): UTF8, UTF16, UTF32. Véase https://es.wikipedia.org/wiki/Unicode

Python 3 utiliza internamente el tipo de datos `str` para representar cadenas de texto Unicode, lo que significa que se puede escribir y manipular texto en cualquier idioma sin preocuparte por la codificación. La compatibilidad con UTF-8 en Python significa que se puede leer, escribir y manipular archivos de texto en cualquier idioma, y también trabajar con datos provenientes de fuentes diversas, como bases de datos, API web, etc., que pueden contener texto en diferentes idiomas y codificaciones.

A continuación se muestran algunos ejemplos.

```
compañero = 'Luismi' # puedo usar la ñ como parte del nombre de la variable
print(compañero)
```

```
Luismi
```

Los códigos Unicode de cada caracter se pueden dar en decimal o hexadecimal, por ejemplo para el símbolo π se tiene el código decimal `120587` y hexadecimal `0x1D70B`. La función `chr()` convierte ese código en el caracter correspondiente:

```
chr(0x1D70B)
```

```
'𝜋'
```

```
chr(120587)
```

```
'π'
```

La función `ord()` obtiene el código Unicode de un caracter y lo regresa en decimal:

```
ord('π')
```

```
120587
```

Podemos usar la función `print()` para realizar una impresión con formato como sigue:

```
π = 3.141592
print('{:04d} \t {} = {}'.format(ord('π'), 'π', π)) # Impresión en decimal
print('{:04o} \t {} = {}'.format(ord('π'), 'π', π)) # Impresión en octal
print('{:04x} \t {} = {}'.format(ord('π'), 'π', π)) # Impresión en hexadecimal
```

```
120587   π = 3.141592
353413   π = 3.141592
1d70b    π = 3.141592
```

Podemos usar acentos:

```
México = 'El ombligo de la luna'
print(México)
```

```
El ombligo de la luna
```

Puedo saber el tipo de codificación que usa Python de la siguiente manera:

```
import sys
sys.stdout.encoding
```

```
'UTF-8'
```

También es posible obtener más información de los códigos unicode como sigue:

```
import unicodedata

u = chr(233) + chr(0x0bf2) + chr(6000) + chr(13231)
print('cadena : ', u)
print()
for i, c in enumerate(u):
    print('{} {:>5x} {:>3}'.format(c, ord(c), unicodedata.category(c)), end=" ")
    print(unicodedata.name(c))
```

```
cadena :  é௲㗯㌯
```

```
é    e9  Ll LATIN SMALL LETTER E WITH ACUTE
௲   bf2  No TAMIL NUMBER ONE THOUSAND
ᜎ  1770  Lo TAGBANWA LETTER SA
㎯  33af  So SQUARE RAD OVER S SQUARED
```

Véase: https://docs.python.org/3/howto/unicode.html

1.2 Asignación múltiple.

Es posible definir varias variables en una sola instrucción:

```
x = y = z = 25
```

```
print(type(x), type(y), type(z))
```

```
<class 'int'> <class 'int'> <class 'int'>
```

```
print(id(x), id(y), id(z))
```

```
94157725269672 94157725269672 94157725269672
```

Observa que se creó el objeto `25` de tipo `<class 'int'>` y los nombres `x`, `y` y `z` son etiquetas al mismo objeto, como se verifica imprimiendo el identificador de cada variable usando la función `id()`.

Podemos eliminar la etiqueta `x` con la función `del()`:

```
del(x)
```

Ahora ya no es posible hacer referencia al objeto `25` usando `x`:

```
print(x)
```

```
NameError: name 'x' is not defined
```

Pero si es posible hacer referencia al objeto `25` con los nombres `y` y `z`:

```
print(y,z)
```

```
25 25
```

Podemos hacer una asignación múltiples de objetos diferentes a variables diferentes:

```
x, y, z = 'eje x', 3.141592, 50
```

```
print(type(x), type(y), type(z))
```

```
<class 'str'> <class 'float'> <class 'int'>
```

```
print(id(x), id(y), id(z))
```

```
139672217233008 139672517638832 94157725270472
```

Como se observa, ahora las variables `x`, `y` y `z` hacen referencia a diferentes objetos, de distinto tipo.

1.2.1 Ejemplos de nombres NO válidos.

Los siguientes son ejemplos NO VALIDOS para el nombre de variables. Al ejecutar las celdas se obtendrá un error en cada una de ellas.

```
1luis = 20      # No se puede iniciar con un número
```

```
SyntaxError: invalid decimal literal (953519616.py, line 1)
```

```
luis$ = 8.2323  # No puede contener caractéres especiales
```

```
SyntaxError: invalid syntax (2653363214.py, line 1)
```

```
for = 35        # Algunos nombres ya están reservados
```

```
SyntaxError: invalid syntax (2521306807.py, line 1)
```

1.3 Palabras reservadas.

Tampoco es posible usar las palabras reservadas para nombrar variables. Podemos conocer las palabras reservadas como sigue:

```
help('keywords')
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
```

```
await       finally     nonlocal    yield
break       for         not
```