Facultad de Ciencias de la Administración

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

PROGRAMACIÓN I

Unidad III – Tipos de Datos

Variables. Expresiones y Operadores.

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración

Universidad Nacional de Entre Ríos

- **Objetivos**
  - Comprender cómo se definen variables y cómo se realizan operaciones con ellas.
  - Entender el concepto de tipo de datos.
  - Identificar las diferencias entre los tipos de datos elementales.
  - Inspeccionar cómo se representa en memoria cada tipo.
- **Temas a desarrollar:**
  - Variables. Definición. Alcance.
  - Expresiones. Operadores lógicos. Operadores aritméticos. Operadores de cadenas de texto. Precedencia.
  - Tipos de datos elementales: Entero, punto flotante, booleano, cadena de texto.
  - Conversión de tipos.
  - Representación de datos en memoria.

Variables

- Una de las características más poderosas de los lenguajes de programación es la habilidad de manipular **variables**.
- *Una **variable** es un **nombre** que se refiere a un **valor**. Al **nombrar valores guardados en memoria**, el desarrollador puede hacer referencia a ellos no solo cuando son creados, sino más adelante en el programa.*
- La variables también permiten manipular valores.
- **Sentencias de asignación**
  - Crea una nueva variable y le da un valor:

```
mensaje = 'Y ahora… algo completamente diferente'

n = 17

pi = 3.1415926535897932
```

- Este ejemplo hace tres asignaciones. La primera asigna un string a una nueva variable con nombre mensaje, el segundo le asigna **17** a `n` y el tercero el (aproximado) valor de π a `pi`.

Nombres de variables

- Los programadores generalmente **eligen** nombres para sus variables que tengan **algún significado** y que muestren **para qué se usan esas variables**.
  - Los nombres de las variables pueden ser de cualquier longitud.
  - Pueden contener letras y números pero no pueden comenzar con un número.
  - Está permitido utilizar letras en mayúscula pero es convención utilizar solo letras en minúscula.
  - El símbolo infraguión _, puede aparecer en el nombre. Es común que se use en nombres con múltiples palabras, tales como **`tu_nombre`**, o **`velocidad_auto`**.
- Si se le asigna un nombre inválido a una variable obtendremos un error de nombre ilegal.
  - `» 10messi = 'Lionel'`
  - `SyntaxError: invalid syntax`
  - `» martin@francisconi = 'martin'`
  - `SyntaxError: invalid syntax`
  - `» class = 'Programación1'`
  - `SyntaxError: invalid syntax`

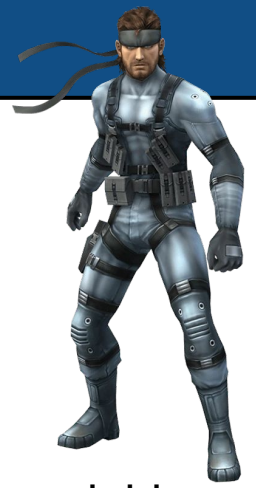
Palabras reservadas

- La palabra `class` es una de las palabras reservadas de **Python**. El intérprete utiliza palabras reservadas para reconocer la estructura del programa, y no pueden ser utilizadas como nombres de variables.

`False`	`class`	`finally`	`is`	`return`
`None`	`continue`	`for`	`lambda`	`try`
`True`	`def`	`from`	`nonlocal`	`while`
`and`	`del`	`global`	`not`	`with`
`as`	`elif`	`if`	`or`	`yield`
`assert`	`else`	`import`	`pass`	`break`
`except`	`in`	`raise`		

- ***No es necesario memorizar el listado de palabras reservadas***. La mayoría de los IDE nos muestran en un color diferente las palabras reservadas.
- Si intentamos utilizar una palabra reservada como nombre de variable nos vamos a enterar!

Nombres de variables (2)



- Algunas cuestiones importantes para tener en cuenta:
  - **Python** es sensible a mayúsculas y minúsculas. Por ejemplo, tanto:
  - **`Variable_Uno`** y **`variable_Uno`**
  - Son identificadas como variables diferentes.
- Cuando elegimos un nombre para nuestros archivos… tampoco es recomendable usar palabras reservadas.
- Para nombrar variables en **Python** se utiliza la sintaxis de los identificadores y la convención de usar ***snake_case*** la cual define que los nombres deben estar en su forma minúscula o mayúscula (dependiendo de si se tratan de variables locales o constantes, respectivamente) y unidos por un '_' o infraguión.
- Pueden contener números pero siempre tienen que empezar con una letra.

Expresiones y sentencias

- Una ***expresión*** es una combinación de valores, variables y operadores. Un valor es por si mismo considerado una expresión y también una variable. Todas las siguientes son expresiones válidas:
  - `» 42`
  - `42`
  - `» n`
  - `17`
  - `» n + 25`
  - `42`
- Cuando se escribe una expresión en el prompt el intérprete la evalúa lo que significa que encuentra el valor de la expresión. Por ejemplo, `n` tiene valor `17` y `n + 25` tiene el valor `42`.
- Una sentencia es una unidad de código que tiene un efecto, como crear una variable o mostrar un valor.
  - `» n = 17 # asigna un valor a n`
  - `» print(n) # muestra por pantalla el valor de n`
- Cuando escribimos una sentencia el intérprete la ejecuta. Es decir, hace lo que dice la sentencia.

Orden de operaciones

- Cuando una expresión contiene más de un operador, el orden de evaluación, depende del orden de las operaciones.
- Para los operadores matemáticos, **Python** sigue las convenciones matemáticas. El acrónimo **PEMDAS** es una forma útil de recordar estas reglas:
  - **Paréntesis:** tiene el más alto nivel de precedencia y pueden ser utilizados para forzar a que una expresión se evalúe en el orden deseado. Dado que las expresiones en paréntesis se evalúan primero, `2 * (3 - 1)` es `4` y `(1 + 1)**(5 - 2)` es `8`. Se puede también utilizar paréntesis para facilitar la lectura como en (minuto * 100) / 60, incluso si la intención no es cambiar la precedencia.
  - **Exponenciación:** tiene el siguiente nivel de precedencia, así que `1 + 2**3` es `9`, no `27` y `2 * 3**2` es `18` no `36`.
  - **Multiplicación y División:** tienen mayor nivel de precedencia que adición y sustracción. Así que `2 * 3 - 1` es `5`, no `4` y `6 + 4 / 2` es `8` no `5`.
  - **Operadores con igual nivel de precedencia:** son evaluados de izquierda a derecha (excepto exponenciación): `grados / 2 * pi`, la división sucede primero y el resultado es multiplicado por `pi`. Para dividir `2` por `pi` se puede usar paréntesis o escribir: `grados / 2 / pi`.
- No es difícil recordar la precedencia de los operadores. Si no podemos darnos cuenta con simplemente mirar el código de la expresión, entonces deberíamos usar paréntesis para que quede más claro.

Operaciones con strings

- En general no se pueden llevar a cabo operaciones matemáticas sobre strings, incluso si los strings parecen números, las siguientes expresiones son inválidas:
  - `'asado' - 'fútbol'` `'pizza' / 'cerveza'` `'fresco' * 'batata'`
- Pero las dos excepciones son + y *.
- El operador **+** lleva a cabo la operación de ***concatenación***, esto significa que dos strings se juntan extremo con extremo. Por ejemplo:
  - `» primero = 'Corona'`
  - `» segundo = 'virus'`
  - `» primero + segundo`
  - `Coronavirus`
- El operador * puede funcionar también en strings para indicar ***repetición*** si uno de los operandos es un **string** y el otro un **entero**. Por ejemplo: `'Tock'*3` es `'TockTockTock'`.
- El uso de **+** y * tiene la misma analogía que suma y multiplicación. Así como `4*3` es el equivalente a `4+4+4` esperamos que `'Tock'*3` sea `'Tock' + 'Tock' + 'Tock'`.
- ¿Hay alguna propiedad que la sumatoria no comparta con la concatenación de strings?

Depuración de programas - Debugging

- Todos los programadores cometemos errores. En la jerga se conocen con el nombre de **bugs** (insectos/bichos).
- Encontrar los errores hasta lograr que nuestros programas hagan lo que queremos nos va a hacer sentir enojados, abatidos o avergonzados. Tenemos que prepararnos para estas situaciones.
- Un enfoque que puede ayudar es pensar en las computadoras como nuestras empleadas que como todas las personas tiene ciertas **fortalezas** y **debilidades**.
  - **Fortalezas**: velocidad y precisión.
  - **Debilidades**: falta de empatía e incapacidad de tener una visión general del problema o tarea encomendada.
- Si pensamos que somos el jefe… nuestra tarea será potenciar esas habilidades y mitigar las debilidades y por otra parte encontrar la forma de comprometernos con el problema sin que nuestras reacciones no interfieran con nuestra habilidad de trabajar.
- Aprender a depurar puede ser frustrante pero es una habilidad valorada y muy útil en muchas actividades más allá de la programación.

Depuración de programas – Debugging (2)

- Hay tres tipos de errores que pueden ocurrir en un programa. Es necesario distinguirlos para rastrearlos más rápidamente:
- **Error de sintaxis: "sintaxis"** hace referencia a la estructura del programa y las reglas sobre la estructura. Por ejemplo, los paréntesis deben coincidir en apertura y cierre. Por eso (1 +2) es correcto pero 8) no.
  - Si se presenta algún error de sintaxis el programa no puede continuar. Durante nuestra primera etapa como programadores vamos a pasar mucho tiempo tratando con errores de sintaxis. Luego con más experiencia se cometerán menos errores y los encontraremos más rápido.
- **Errores de tiempo de ejecución:** El segundo tipo de error **no aparece hasta que el programa empieza a ejecutarse**. Estos errores también se llaman ***excepciones*** porque comúnmente indican que algo excepcional ha sucedido.
  - Los errores en tiempo de ejecución son extraños en programas simples así que nos los vamos a encontrar más adelante en el curso.
- **Errores de semántica:** Si hay algún error de semántica, nuestro programa **va a ejecutarse sin arrojar ningún mensaje de error** pero **no va a hacer lo correcto**. **Va a hacer otra cosa**. Específicamente, va a hacer lo que le indicamos.
  - Identificar errores de semántica puede ser difícil porque hay que buscar hacia atrás mirando los resultados del programa e intentando identificar qué es lo que está haciendo.

Bibliografía

- Óscar Ramírez Jiménez: ***"Python a fondo"*** 1era Edición. Ed. Marcombo S.L.. 2021.
- Allen Downey. ***"Think Python"***. 2Da Edición. Green Tea Press. 2015.
- Eirc Matthes: ***"Python Crash Course"***. 1era Edición. Ed. No Starch Press. 2016.
- Zed A. Shaw: ***"Learn Python 3 the Hard Way"***. 1era Edición. Ed. Addison-Wesley. 2017.