Unidad 1: Introducción a Estructuras de Datos

Estructuras de Datos

Facultad de Ciencias de la Administración - UNER

In [1]:
```
from datetime import datetime
print(f"{'PRIMER' if datetime.now().month < 7 else 'SEGUNDO'} CUATRIMESTRE DE {datetime.now()
```

```
PRIMER CUATRIMESTRE DE 2024 😎
```

Objetivos

- Conocer las principales características de **Python**.
- Entender cómo definir variables y realizar operaciones con ellas.
- Conocer los tipos de datos disponibles en el lenguaje.

Introducción

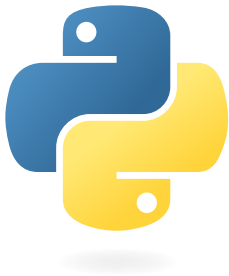
- Durante este curso aprenderemos a programar haciendo uso de **Tipos** y **Estructuras de Datos**, obteniendo así, soluciones más simples y/o eficientes a los problemas que se nos planteen.
- Los **Tipos de Datos** que definamos, serán producto de la formalización en un lenguaje de programación, de conceptos que abstraemos de la realidad.
- La finalidad de esta clase es que los estudiantes tengan un primer contacto con el lenguaje de programación en **Python** y comprendan sus características principales.

Python - Origen

- **Python** es un lenguaje de programación interpretado de alto nivel.
- En 1989, **Guido van Rossumm** (holandés) comenzó como hobby el desarrollo de **Python**, con el objetivo de mejorar la interfaz de usuario del Sistema Operativo Amoeba.
- En un principio, se pensó como un lenguaje de programación pequeño para utilizar en **CWI** (Centrum Wiskunde & Informatica) incorporando características adicionales que ayudaran a interactuar mejor con el sistema operativo.
- La primera versión de **Python** fue lanzada en febrero de 1991 con el número de versión 0.9.0.
- El nombre del lenguaje proviene de la afición que tenía van Rossum a la serie de televisión *Monty Python´s Flying Circus* y no de algo relacionado con el mundo de los reptiles.

Python - Definición

- **Python** es un lenguaje de alto nivel, de propósito general, multiparadigma principalmente imperativo, orientado a objetos y funcional, de tipado dinámico y fuertemente tipado a nivel de lenguaje de programación.



Características - Ventajas (1)

- Sus principales ventajas de **Python** son:
  - **Sintaxis sencilla, simple y clara**: permite desarrollar programas de forma intuitiva. Leer un programa escrito en **Python** es muy parecido a leer un texto anglosajón.
  - La claridad viene de la mano de:
    - Los bloque lógicos se definen utilizando identación en vez de `{` y `}` (usado en lenguajes como **Java**, **JavaScript** o **C**).
    - Las expresiones simples no necesitan el uso de paréntesis (como **JavaScript**).
    - Para la separación de instrucciones se utiliza el salto de línea ↩ en vez del carácter `;`.
    - Posee un sistema de recolección de basura 🗑.

Características – Ventajas (2)

- **Interpretado**: no es necesario compilar los programas cada vez que se hace un cambio en el código.
  - Representa una gran ventaja frente a los lenguajes compilados (**C, C++, Java**) y aumenta considerablemente la velocidad de desarrollo de aplicaciones.
  - Por otro lado, el código es **independiente del hardware** en el que se ejecuta resultando en programas multiplataforma gracias al uso de su máquina virtual.

Características – Ventajas (3)

- **Baterías incluidas**: posee multitud de herramientas en la biblioteca estándar que ayudan a realizar un sinfín de aplicaciones sin necesidad de utilizar bibliotecas de terceros.
  - También permite la integración con otros lenguajes de programación. Así, podemos en **C, C++, .Net o Java** ejecutar código **Python**, y viceversa. Usando diferentes técnicas, el código **Python** se puede transcompilar en otro lenguaje (como **JavaScript**) u otros lenguajes pueden ejecutar código **Python** haciendo uso de subprocesos u otras técnicas.

Características – Ventajas (4)

- **Multiplataforma**: se puede ejecutar sobre PCs con Windows, Linux o Mac OSX, así como también en otros dispositivos electrónicos como teléfonos, relojes inteligentes y consolas de videojuegos.
- **Libre, de código abierto y gratuito**: puede ser usado, copiado, estudiado, y modificado de cualquier forma.
  - El código de **Python** es compartido libremente y se alienta a la comunidad de programadores que mejoren el diseño del software.
  - Las distintas versiones de **Python** pueden descargarse desde el sitio Web oficial: [Enlace].
- **Comunidad**: **Python** cuenta con extensa documentación y una enorme comunidad.
  - La comunidad oficial **"Python Software Foundation" (PSF)**: [Enlace].
  - La evolución de **Python** se rige por la aprobación/implementación de propuestas de mejora conocidas con el nombre de **PEP** siguiendo un proceso de aprobación que participa e involucra a los miembros de la comunidad.

Características - Debilidades

- Al igual que cualquier otro lenguaje de programación, **Python** también tiene puntos débiles:
  - **Lentitud**: en muchas ocasiones se considera que los programas desarrollados en **Python** son **"lentos"** en comparación con los tiempos de ejecución de lenguajes compilados.
    - El origen de esta afirmación reside en que se trata de **lenguaje interpretado** y no poseer por defecto un compilador JIT (del inglés Just in – Time), lo que haría que se compilase el programa escrito en **Python** y optimizasen más los tipos de datos.
    - Aún así, en **Python 3** se han hecho notables mejoras de rendimiento en los tipos de datos.
    - No obstante, existen librerías que permiten marcar porciones de código para ser compiladas en tiempo de ejecución o la opción de utilizar **CPython**, que permite escribir código **C** compatible con **Python** e integrarlo de forma natural para mejorar la velocidad de procesamiento.

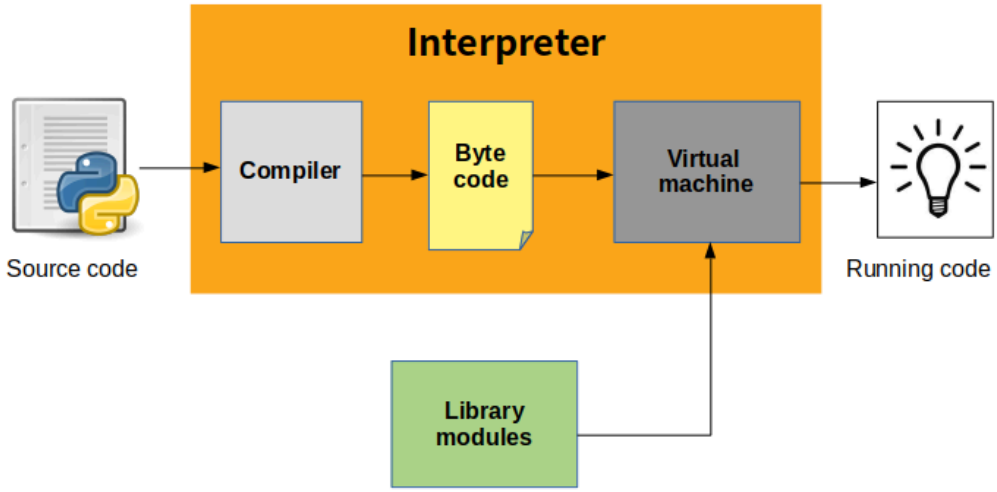
Versiones

- A lo largo de su historia **Python** ha sufrido numerosos cambios y hoy en día sigue recibiéndolos continuamente a través de las **PEP (Python Enhancement Proposal)**.
- A continuación, se nombran las principales versiones con los cambios más destacados:
  - Versión 0.9 (1991): la primera versión publicada por van Rossum. Contaba con muchos componentes actuales como listas, diccionarios, conceptos de orientación a objetos, cadenas de caracteres, entre otros.

- Versión 1.0 (1994): se introducen conceptos de programación funcional.
- Versión 1.6 (2000): se añade licencia compatible con GPL (GNU General Public License).
- **Versión 2.7 (2010)**: última versión de la rama 2.X. En 2014 se anuncia que la versión 2.X dejará de tener soporte a principios de 2020, invitando a los usuarios a migrar activamente a la versión 3.
- **Versión 3.0 (2008)**: se hacen cambios en cuestiones principales del lenguaje, quitando redundancia de código e introduciendo grandes incompatibilidades con la versión 2.
- Versión 3.9 (2020): se añaden múltiples funcionalidades y se borran algunas presentes por retrocompatibilidad con la versión 2. Desde aquí en adelante no funciona en Windows 7.
- **Versión 3.12.2** (06/02/2024): última versión del lenguaje.

Intérprete de Python

- El código fuente de **Python** no necesita ser compilado a código máquina específico del hardware donde se ejecuta. La ejecución se realiza en cualquier sistema que tenga instalada la **Máquina Virtual de Python**.
- Instalar **Python** viene con dos componentes fundamentales:
  - El **intérprete** y
  - la **biblioteca estándar** (módulo, funciones, constantes, tipos, tipos de datos, excepciones, etc.).



Fuente: https://python-kurs.github.io/sommersemester_2019/units/S01E01.html

Intérprete de Python (2)

- Se podría definir a un intérprete como un programa que se encarga de ejecutar otros programas.
- A continuación se ahondará en ello:
  - El código fuente, son archivos de texto plano que tienen una gramática específica (*lenguaje Python*), con una extensión concreta (**.py**) y estructurados de una forma precisa.
  - Los archivos **byte code**, son el resultado de una compilación rápida que se efectúa justo antes de comenzar la ejecución. El código escrito en **byte code** está listo para ser ejecutado en cualquier **Máquina Virtual de Python**.
  - Por último, la **Máquina Virtual de Python (PVM – Python Virtual Machine)**, es la encargada de ejecutar los archivos que tienen el **byte code**. Por lo tanto, la parte que sí es dependiente del hardware utilizado es la **Máquina Virtual**.

- Lo que se denomina **intérprete de Python** es el programa completo que analiza el código fuente, genera los ficheros compilados y ejecuta el código usando la máquina virtual.

Intérprete de Python (3)

- Cabe destacar algunas peculiaridades del **byte code**:
  - Los archivos que contienen el byte code tienen una extensión **.pyc (Python compiled)**.
  - Estos archivos no son necesarios para la ejecución del programa, dado que, si no se pueden generar por algún motivo (por falta de espacio o de permisos de escritura), el byte code será generado e insertado en memoria directamente, sin crearse en ficheros guardados en el sistema operativo.
  - Un programa en **Python** que tenga los archivos **.pyc** generados no necesita tener el código fuente, por lo que se puede ahorrar espacio de disco borrando los códigos fuente solo ejecutando los **.pyc.** (solo se recomienda en sistemas con grandes restricciones de espacio)
  - El **intérprete de Python** es inteligente a la hora de generar los **.pyc**, si ya se han generado los ficheros .pyc y no ha habido cambios en el código fuente, no realiza ninguna compilación, simplemente usa los archivos ya compilados, agilizando así el proceso de iniciar la aplicación.

Implementaciones

- Cuando se habla de la implementación de **Python**, normalmente se hace referencia a la implementación basada en **C** denominada **CPython**.
  - No obstante, el intérprete puede ser implementado en otros lenguajes. Las diferencias entre implementaciones están, principalmente, en la habilidad de usar librerías escritas en algún lenguaje específico.
- **CPython**:
  - Es la implementación original del lenguaje y la más utilizada, programada en ANSI C.
  - Si un sistema operativo tiene **Python** preinstalado, lo más seguro es que se trate de **CPython**.
  - Es el estándar, siempre se mantiene actualizada, soporta la interoperabilidad con librerías escritas en **C** y normalmente es muy rápida en tiempo de ejecución comparada con las demás implementaciones. En GitHub: https://github.com/python/cpython
  - Otras implementaciones: **Jython, PyPy, IronPython**.

Hola Mundo! en Python

- Es tradición, que nuestro programa cuando aprendemos un nuevo lenguaje de programación se llame "Hola Mundo!".
- En **Python** se hace así:

```
In [2]: print("Hola Mundo!")
```

```
Hola Mundo!
```

- Los paréntesis indican que `print` es una función.
- Podemos usar indistintamente comillas simples o dobles, obteniendo el mismo resultado: `print ('Hola Mundo!')`
- Definimos comentarios con el símbolo `#`. Todo lo que se encuentre a la derecha de este símbolo será considerado como un comentario. Ejemplo: `60 * 60 * 1000 # la cantidad de`

`milisegundos en una hora`

Variables

- A diferencia de otros lenguajes de programación, en **Python, las variables son creadas cuando aparecen por primera vez a la izquierda de una operación de asignación.**

In [3]:
```
nombre = 'Eddie Munson'
print(nombre)
```

```
Eddie Munson
```

- Con respecto a los nombres de variables tenemos que saber que:
  - 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 minúsculas.
  - 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.

In [4]:
```
messi = 'Lionel'
nachonovello = 'nacho'
clazz = 'Programación1'
```

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
None	continue	for	lambda
True	def	from	nonlocal
and	del	global	not
as	elif	if	or
assert	else	import	pass
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.

Nombres de variables (2)

- Algunas cuestiones importantes para tener en cuenta:
  - **Python** es sensible a mayúsculas y minúsculas. Por 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 `_` (infraguión).
- Pueden contener números pero siempre tienen que empezar con una letra.

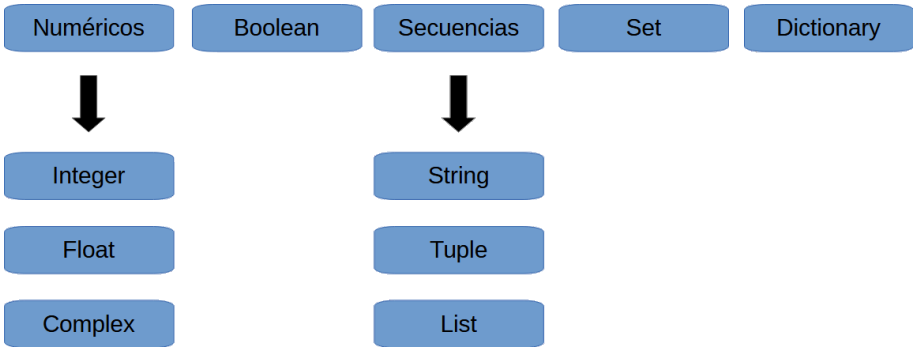
Tipos de datos

- **Python** provee un amplio abanico de tipos de datos fácilmente instanciables e intuitivos que permiten que cualquier desarrollador pueda enfocarse en qué construir y no en cómo hacerlo.

Tipos de datos (2)

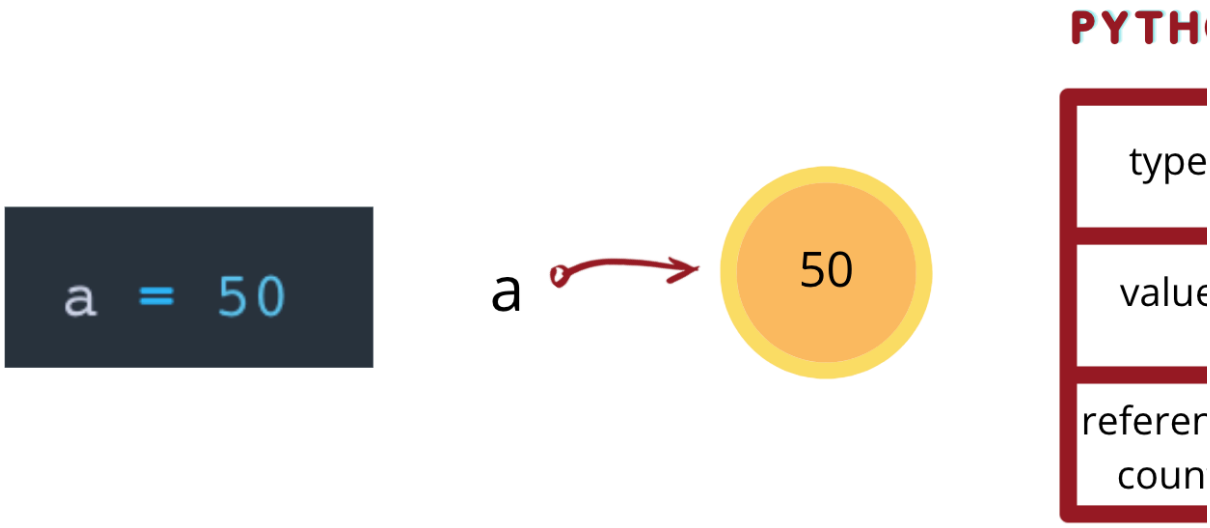
Grupo	Nombre	Tipo	Ejemplo
Numéricos	Entero	`int`	`34, 1_999, -12, -98`
	Punto flotante	`float`	`1.62, 5.7e8`
	Complejo	`complex`	`5j, 2 + 8j`
Secuencias	Listas	`list`	`[1, 2, 3]` `[3.14, False, 'c']`
	Tuplas	`tuple`	`(3, 4, True)`
	Secuencias numéricas	`range`	`range(5)`
Secuencias de texto	Cadenas de caracteres	`str`	`'casa', "color", ''tecla'', """gato"""`
Secuencias Binarias	Cadenas binarias	`bytes`	`b'coche'`
	Cadenas binarias mutables	`bytearray`	`bytearray(b'hola')`
Conjuntos	Conjunto	`set`	`set([3, true, 2]), {4, False, 12}`
	Conjunto estático	`frozenset`	`frozenset([2, 'hola', True, 3])`
Mapas	Diccionarios	`dict`	`{'x': 1, 'y': 2}, dict(x=90, y=20)`

Tipos de datos (3)



Literales, variables y tipos de datos (2)

- En **Python** los valores son objetos y es el propio intérprete el que se encarga de saber dónde se ubican en la memoria y cómo acceder a ellos.
- Por este motivo, a la hora de desarrollar aplicaciones no es necesario dedicarle mucho esfuerzo a la gestión de memoria.
- Conceptualmente los objetos en **Python** se pueden ver como dos componentes:
  - Una **referencia**, que guarda la dirección de memoria en la que está alojado el dato.
  - Un **contenedor** que representa el almacenamiento del objeto guardado en sí, el cual contiene información relevante como el tipo de dato y la información que alberga.



Variables, identificadores y tipos de datos

- Podemos conocer la dirección de memoria donde se encuentra un objeto utilizando la función `id()`:

In [5]:
```
print("id(2):", id(2))
var1 = "Soy var1!"
print("id(var1):", id(var1))
print("id('Hola Mundo!'):", id('Hola Mundo!'))
```

```
id(2): 140720378602312
id(var1): 2543395316016
id('Hola Mundo!'): 2543395316912
```

- Si dos valores tienen el mismo identificador, entonces, ambos valores son exactamente el mismo.
- Si bien esto permite ahorrar espacio en la memoria puede conllevar un inconveniente:
  - Si se cambia el valor de referencia del identificador, el cambio afectará a todos los objetos que lo usen.
- Para conocer el tipo de datos de una variable o valor podemos utilizar la función `type()`.

In [6]:
```
print("type(2):", type(2))
print("type(42.0):", type(42.0))
print("type('Hola Mundo!')", type('Hola Mundo!'))
```

```
type(2): <class 'int'>
type(42.0): <class 'float'>
type('Hola Mundo!') <class 'str'>
```

Tipos Booleanos

- Los conceptos de **verdadero** y **falso** están presentes y modelados como **booleanos** con dos valores constantes: `True` y `False` (con la primera letra en mayúscula).
- La función `bool()` permite convertir cualquier valor en un valor booleano:

In [7]:
```
print(True, False)
print(bool(True), bool(False))
print(bool(0), bool(0j), bool(''), bool(None), bool(set()))
print(bool(1), bool(-1), bool('casa'), bool(24))
```

```
True False
True False
False False False False False
True True True True
```

- Todos los valores en **Python** tienen asociada una noción de verdad. Algunos son evaluados como falso:
- Las constantes `None` y `False`
- Los valores numéricos interpretados por cero: `0, 0.0, 0j`
- Los objetos vacíos: `''`, `""`, `()`, `dict()`, `set()`, `range(0)`, `[]`

Tipos Booleanos - Operadores

- Las operaciones lógicas con booleanos son tres: `or`, `and` y `not`, como se muestra en la siguiente tabla:

Operador	Ejemplo	Resultad
or	» x or y	Si x es False, entonces y,
and	» x and y	Si x es False, entonces x;
not	» not x	Si x es False, entonces T False

Tipos Booleanos – Operadores – Cortocircuito lógico

- El cortocircuito lógico es una propiedad que implementa **Python** para la evaluación de expresiones y que, además, ayuda a hacerlas más eficientes puesto que no necesita evaluar las expresiones completas.
- Durante la evaluación de una expresión and se encuentra un elemento que es False, se detiene la ejecución (cortocircuita la ejecución) y devuelve ese valor.
- Hace lo mismo con las expresiones `or`, pero con los elementos que devuelvan `True`.

Tipos Numéricos

- En el núcleo de **Python** existen tres tipos numéricos definidos que permiten expresar los números enteros, los números de coma flotante (números reales) y los números complejos en forma sencilla, así como operar con ellos.
- **Operadores**:
  - Existen operaciones numéricas compartidas por todos los tipos numéricos de **Python**, Además, se pueden combinar valores de tipos numéricos diferentes en cuyo caso prevalecerá el más general:

Operador	Descripción	Ejemplo
+	Suma	5 + 8
-	Sustracción	90 - 10
*	Multiplicación	4 * 7
/	División tradicional	7 / 2
//	División entera	7 // 2
%	Módulo	7 % 3
**	Exponenciación	3 ** 4

Enteros

- Los enteros son valores numéricos más simples e intuitivos son del tipo `int`. Ejemplos:

In [8]:
```
print(-1)
print(100)
print(+123)
print(5_000_000)
print(05) # la expresión: 05 (anteponiendo cero) arroja error de sintaxis.
```

```
  Cell In[8], line 5
    print(05) # la expresión: 05 (anteponiendo cero) arroja error de sintaxis.
          ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0o prefix for octal integers
```

- En **Python** los enteros pueden ser tan grandes como sea necesario, dado que no tienen un número máximo fijo asignado.
- La función `int()` toma un argumento de entrada y nos devuelve un valor entero equivalente.

In [ ]:
```
print(int(True))
print(int(False))
```

- La conversión de números de punto flotante retorna un entero perdiéndose la parte fraccionaria.

In [ ]:
```
print(int(4.8))
```

Números de punto flotante

- Los números de punto o coma flotante (o números reales) forman parte del conjunto de tipos numéricos implementado en **Python** y permite realizar operaciones de forma fácil y sencilla gracias a las operaciones que hay disponibles en el núcleo. Ejemplos:

In [ ]:
```
print(5.)
print(5.0)
print(05.0)
# En notación científica:
print(5e0)
print(5.0e1)
print(5.0e10)
```

- La función que nos permite convertir cualquier valor en un número de punto flotante es `float()`.

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`.
- Es fácil 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:

In [ ]:
```
print('asado' -  'fútbol')
print('pizza' / 'cerveza')
print('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. Ejemplo:

In [ ]:
```
primero = "Corona"
segundo = "virus"
print(primero + segundo)
```

- El operador `*` puede funcionar también en strings para indicar repetición si uno de los operandos es un string y el otro un entero. Ejemplo:

In [ ]:
```
print("Tock" * 3)
```

- 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"`.

Comparaciones

- En **Python** los objetos se pueden comprar entre sí con los ocho tipos de operadores básicos:

Operador	Ejemplo	Descripción
>	» x > y	x es mayor que y
>=	» x >= y	x es mayor o igual que y
<	» x < y	x es menor que y
<=	» x <=	X es menor o igual que y
==	» x == y	X es igual a y
!=	» x != y	X es distinto de y
is	» x is y	X es un objeto idéntico a y
is not	» x is not y	X no es un objeto idéntico a

Bibliografía

- Pablo A. García, Marcelo A. Haberman, Federico N. Guerrero: "*Programación E1201: curso de grado*". 1era Edición. Editorial de la UNLP. 2021.
- Óscar Ramírez Jiménez: "*Python a fondo*" 1era Edición. Editorial Marcombo S.L.. 2021.
- Allen Downey. "*Think Python*". 2da Edición. Editorial Green Tea Press. 2015.
- Eirc Matthes: "*Python Crash Course*". 1era Edición. Editorial No Starch Press. 2016.
- Zed A. Shaw: "*Learn Python 3 the Hard Way*". 1era Edición. Editorial Addison-Wesley. 2017.