

▾ Clase 1: Expresiones y Tipos de Datos Básicos

Objetivos del curso

- Base de razonamiento lógico y algorítmico.
- Modelamiento y abstracción.
- Importante para resolver problemas!

¿Qué es un algoritmo?

Un algoritmo es una secuencia finita de pasos que permiten ejecutar cualquier tarea (como por ejemplo, hacer un huevo frito). La palabra algoritmo viene de la transcripción latina del nombre de *Abu Abdallah Muhammad ibn Musa al-Khwarizmi*, un famoso matemático, astrónomo y geógrafo persa del siglo IX, padre del álgebra y quien introdujo el concepto matemático de algoritmo.

Ejemplo "cocinar un huevo frito".

- Problema?
- Elementos?
- Pasos de ejecución (romper el problema en unidades básicas):
 1. Encender un `fósforo`
 2. Con el `fósforo`, prender un quemador de la `cocina`
 3. Colocar la `sartén` sobre el quemador de la `cocina`
 4. Poner unas gotas de `aceite` sobre la `sartén`
 5. Tomar un `huevo` y quebrarlo
 6. Colocar el huevo quebrado sobre la `sartén`
 7. Esperar hasta que el `huevo` esté listo

¿Qué es un programa?

- Especificación ejecutable de un algoritmo.
 - Para representar un programa en un computador utilizamos un lenguaje de programación (Python)
 - Comenzaremos a trabajar con su interprete: lee las expresiones, las evalúa e imprime su resultado.
 - Python es una herramienta no es un fin, el objetivo es aprender a resolver problemas.
-

Programas simples

Tipos de datos básicos

Indica al computador las características de los datos con los que va a trabajar. Esto incluye imponer restricciones sobre los datos, tales como qué valores puede tomar o qué operaciones se pueden realizar sobre éste. Todos los valores en un programa tienen un tipo.

Enteros (int): `-1, -2, -3, ..., 0, 1, 2, ..`

Reales (float): `0.3456`, cualquier número con punto decimal.

Texto (str): `'hola', "Mi mascota es un ñandú", '123'` etc.

▼ Evaluación de expresiones

Con los tipos de datos explicados anteriormente, podemos realizar operaciones entre ellos utilizando operadores específicos en cada caso. Así, para datos numéricos (enteros y reales), podemos usar los operadores de suma (+), resta (-), multiplicación (*) y división (/). Las prioridades de los operadores es la misma usada en álgebra

```
3+5
```

 8

```
3+2*5
```

 13

```
(3 + 2) * 5
```

 25

Operador de potencia (**):

```
2 ** 3
```

 8

```
4**0.5
```

 2.0


Y calcular el resto de una división entera (%):

```
10%3
```


 1

Operaciones con texto, concatenación (unir o sumar '+') y repetir '*' :

```
'abra' + 'cadabra'
```

 'abracadabra'

```
'ja'*3
```

 'jajaja'

En muchos lenguajes de programación el tipo de dato del resultado de evaluar una expresión dependerá de los operandos. Por ejemplo, en el caso del operador + (suma), si la expresión está compuesta por enteros el resultado también será entero:

```
1+2
```

 3

```
1.0+2.0
```

 3.0

```
1+2.0
```

 3.0

```
1.0+2
```

 3.0

El único operador que no sigue la regla anterior es ' / ' (división), que siempre produce números flotantes:

```
1/2
```

 0.5

Si quiero que la división entregue sólo la parte entera del resultado debo usar el operador ' // ' (división entera):

```
1//2
```

 0

```
1.0//2.0
```


 0.0

Finalmente, si queremos juntar valores de tipo texto con valores de tipo numérico, debemos convertir estos últimos previamente a valores de tipo texto. Por ejemplo, si queremos transformar un valor `n` a un valor equivalente de tipo texto, utilizamos la función de Python `str`. De igual manera, si tenemos un valor de tipo texto que se puede entender como número, por ejemplo `'103'`, podemos convertir el valor en tipo numérico usando las funciones `int` y `float`. Así:

```
'En el curso hay ' + str(30) + ' alumnos'
```

 'En el curso hay 30 alumnos'

```
'100'+'1'
```

 '1001'

```
int('100') + 1
```

 101

▾ Variables

Sirven para guardar resultados de evaluaciones de expresiones.

```
a = 8 # la variable a contiene el valor 8
```

```
b = 12 # la variable b contiene el valor 12
```

```
a # mostramos el valor de a
```

```
a # mostramos el valor de a
```

 8

```
a+b # creamos una expresión y mostramos su valor
```

 20

```
c = a + 2 * b # creamos una expresión, se evalúa y se define c con su valor
```

```
c # mostramos el valor de c
```

 32

```
a = 10 # redefinimos a
```

```
a + b
```

 22

```
c # el valor no cambia ya que lo cambiamos antes de la re-definición de a
```

 32

sirve para introducir comentarios

▼ Usar nombres descriptivos para las variables

Mal ejemplo:

```
a = 8
b = 12
c = a * b
```

Mejor:

```
ancho = 8
largo = 12
area = ancho*largo
area
```

 96

```
dia='12'; mes='marzo'; agno='2018'
hoy = dia + ' de ' + mes + ' de ' + agno
hoy
```

 '12 de marzo de 2018'

▼ Errores

Errores de tipo

```
dia = 13
```

```
mes = 'marzo'
'Hoy es ' + dia + ' de ' + mes
```



```
-----
-
TypeError                                Traceback (most recent call
last)
<ipython-input-31-fadc92769dcc> in <module>()
      1 dia = 13
      2 mes = 'marzo'
----> 3 'Hoy es ' + dia + ' de ' + mes

TypeError: must be str, not int
```

```
dia = 13
mes = 'marzo'
'Hoy es ' + str(dia) + ' de ' + mes
```



```
'Hoy es 13 de marzo'
```

Errores de indentación

```
x = 3
  x
```



```
File "<ipython-input-33-330f11e398a5>", line 2
  x
  ^
IndentationError: unexpected indent
```

SEARCH STACK OVERFLOW

Errores de sintaxis

```
numero = 15
antecesor = (numero - 1))
```



```
File "<ipython-input-34-037f1509520b>", line 2
  antecesor = (numero - 1))
                        ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

Errores de nombre

```
lado1 = 15
area = lado1/lado2
```



```
-----
-
NameError                                Traceback (most recent call
last)
<ipython-input-35-e743b4fefc30> in <module>()
      1 lado1 = 15
----> 2 area = lado1/lado2

NameError: name 'lado2' is not defined
```