Herramientas de programación en Python

1. Fundamentos de programación

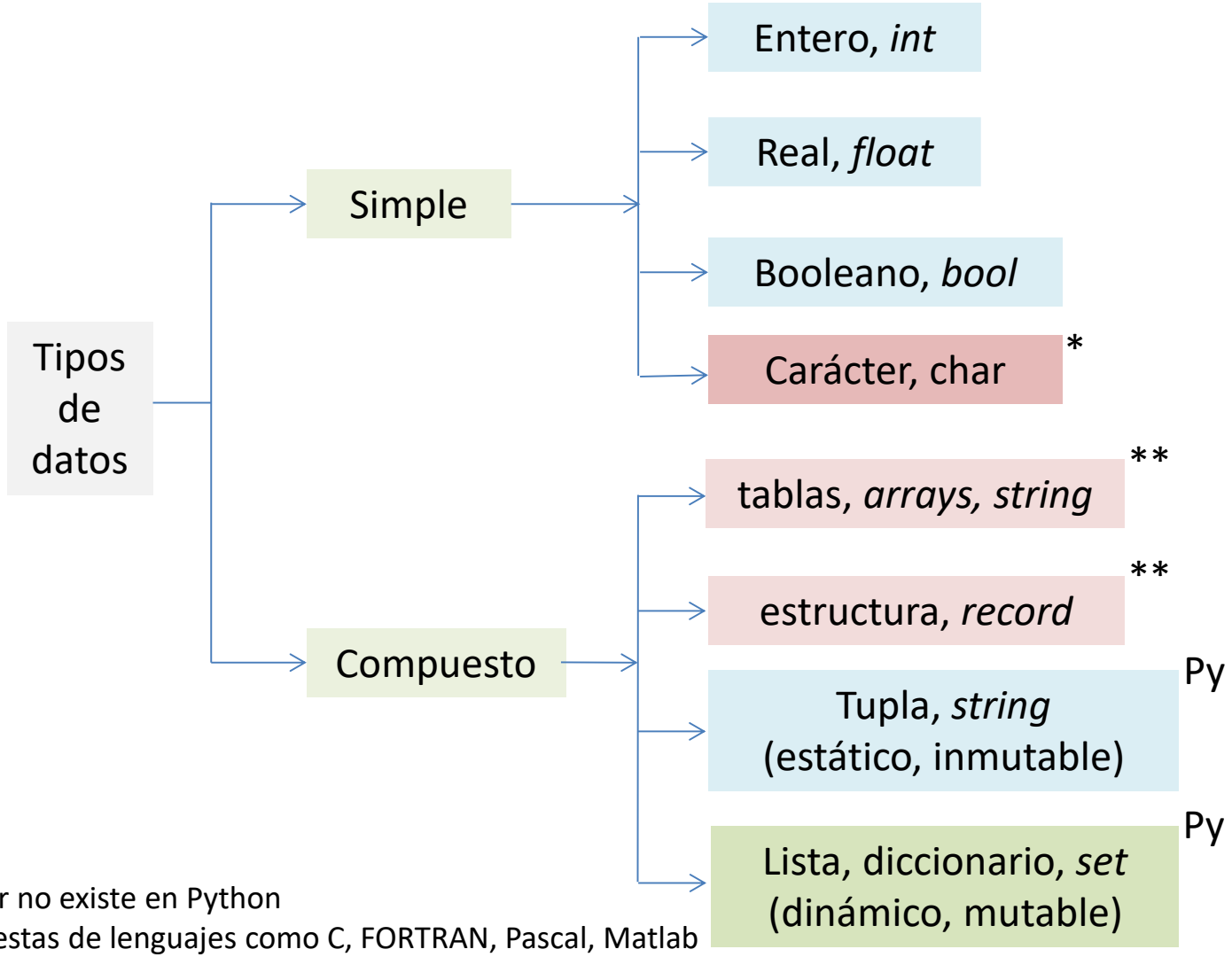
1.1 Tipos de datos simples, objetos, variables y operadores

1. Tipos de datos simples
2. Variables y Expresiones
3. Operaciones
4. Acciones elementales

Pedro Gomis

pedro.gomis@upc.edu

Tipos de datos



* Tipo de dato carácter no existe en Python

** Estructuras compuestas de lenguajes como C, FORTRAN, Pascal, Matlab

Py Estructuras compuestas en Python

Tipos de datos

- **Datos Simples (Escalares)**

  Numéricos:

  Enteros. Ejemplos: -4, 0, 25, 1000

  Reales. Ejemplos: 1.5, -4.33, 123.0

  Caracteres:

  Letras, símbolos, dígitos. Ejemplos: `'a', 'A', 'g', '$', '%', '1' , '.', '4', '*'`

  Lógicos o Booleanos:

  Valores: falso, verdadero (cierto) $\longleftrightarrow$ (0, 1)

- **Datos Compuestos o estructurados**

  Formado por agrupaciones de datos. Ejemplos: string, arrays, tuplas, listas, etc.

Tipos de datos

Enteros (*Integer*)

- Los enteros, en la mayoría de lenguajes de programación incluyendo Python, se definen con la palabra ***int***
- En Python no hay limitación de bits para representarlos

```
>>> type(7)
<class 'int'>
>>> a = 45
>>> type(a)
<class 'int'>
```

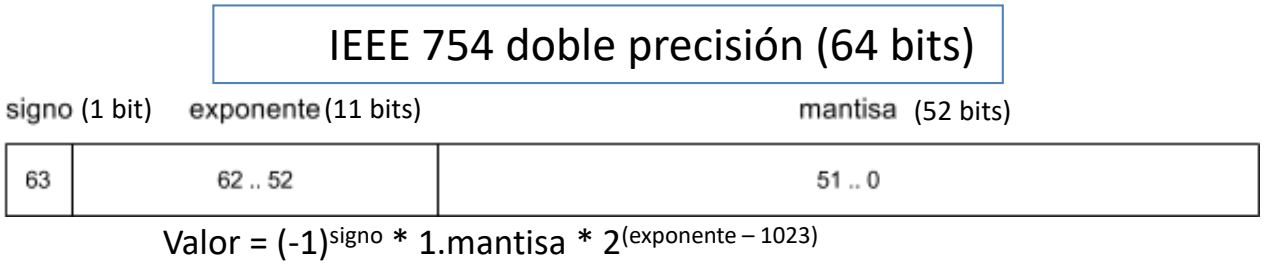
```
>>> bin(5)
'0b101'
>>> bin(200)
'0b11001000'
```

```
>>> 2**220     # 2 elevado a 220
1684996666696914987166688442938726917102321526408785780068975640576
```

Tipos de datos

Reales (*float*)

- Notación de los Números **reales** en Pascal, C, Matlab, Python, etc.
- Cuando se usa el carácter **E** o **e** dentro de un número real representa "veces 10 elevado a". Por ejemplo, el número 4000 se representa por la mantisa 4 y el exponente 3, 4e3. Se lee 4 veces 10 elevado a la 3.




```
>>> 25e-2
0.25
>>> 4e3
4000.0
>>> type(4.)
<class 'float'>
>>> .2e2
20.0
>>> 1.1 + 2.2   # se representa con error en punto flotante binario
3.3000000000000003
```

No todos los números reales pueden representarse de forma exacta

Tipos de datos

Booleano (*Boolean*)

- El tipo de dato para representar valores lógicos o booleanos en Python es ***bool***. En Pascal y C++ se definen como ***boolean*** y ***bool***, respectivamente
- Los datos booleanos toman el valor ***True (1)*** o ***False (0)***.
- El nombre booleano se usa luego que George Boole, matemático inglés, propusiera en el siglo XIX un sistema algebraico basado en estos dos valores lógicos y tres operaciones lógicas: ***or, and not***

```
>>> a = 3 > 2
>>> a
True
>>> type(a)
<class 'bool'>
>>> 4 > 5
False
```

Tipos de datos

Carácter (*Character*)

- El tipo de dato carácter usado en varios lenguajes de programación es el elemento escalar o indivisible de los textos usados en informática.
- Los textos se llaman cadena de caracteres (en inglés ***string***). Los caracteres están ordenados de acuerdo a la tabla ASCII. Los caracteres ASCII ordenados del valor decimal 20 al 127 son:

```
  ! " # $ % & ' ( ) * +, -. / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B
C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d
e f g h i j k l m n o p q r s t u v w x y z { | } ~
```

- El tipo **carácter no está definido en Python**. Los caracteres simples se definen igual que un texto con una sola letra, es decir como una cadena de caracteres (**string**).

```
>>> type('a')
<class 'str'>
```

Tipos de datos

Datos compuestos de cadena de caracteres: *string*

- El tipo de dato ***string*** es la estructura básica para manejar texto, codificación ASCII o UTF-8.
- Los **string** en Python se definen entre comillas simples (' ') o dobles (" "). También se pueden definir entre comillas triples (""") cuando se tengan múltiples líneas.

```
>>> 'Hola'
'Hola'
>>> b = "Casa de madera"
>>> type(b)
<class 'str'>
>>> type(15)
<class 'int'>
>>> type('15')
<class 'str'>
```

```
>>> 'Ella dijo "Qué lindo"'
'Ella dijo "Qué lindo"'
>>> "He doesn't know"
"He doesn't know"
```

Datos compuestos de cadena de caracteres: *string*

- Se puede también usar el carácter barra invertida (\) que sirve de escape para agregar comillas u otras acciones dentro del ***string***

```
>>> print('He doesn\'t know I \"will come\"')
He doesn't know I "will come"
```

- **\n** indica salto a nueva línea. Se pueden incluir múltiples líneas en un string usando triples comillas """ ... """

```
>>> print('Cambiamos de línea \nNueva línea')
Cambiamos de línea
Nueva línea
>>> """
    Programa:
    Autor:
    Fecha:
    """
'\n    Programa:\n    Autor:\n    Fecha:\n    '
```

Variables

- En ***matemáticas*** las variables se usan para representar valores numéricos. Se utiliza un carácter o texto para representarlas. En cálculo matemático una función del tipo ***y*** = *f(**x**)* involucra dos variables, ***x*** e ***y***.
- En los lenguajes de programación se requiere normalmente recordar o ***guardar*** los valores numéricos, booleanos o de texto para ser ***usados*** una o múltiples veces en el programa. ***Las variables tienen este cometido***.
- Intuitivamente, podemos pensar que la **variable** es un lugar en la memoria del computador en la que se almacena la información que vamos a procesar.
- En los lenguajes como el FORTRAN, C/C++ o Pascal, una variable se considera un ***contenedor*** o ***lugar dentro de la memoria RAM*** del computador, con un nombre asociado (***identificador***), donde se guarda un valor de un tipo dado.
- Sin embargo, en **Python *el concepto es algo diferente***, pues las variables no son un lugar de memoria que contienen un valor sino que **se asocian, o refieren, a un lugar de memoria que contiene ese valor**.

Acción de asignación

- La acción de **asignación** se usa para darle a una variable un valor determinado.
- En ***Python la acción de asignación*** de valores a una variable quiere decir que ***la variable*** con su nombre determinado ***se va a asociar al valor de la derecha de la asignación***:

```
>>> a = 7
```

- Asignación en otros lenguajes:

Lenguaje de Programación	Símbolo de asignación
C/C++, FORTRAN, Python, Java	=
Pascal	:=
R	<-

- La sentencia o instrucción `a = 7` crea primero un **objeto** con el valor 7 de tipo **int** y se ubica en un lugar de memoria. Este ***lugar de memoria*** se denomina en Python ***identidad del objeto***

Acción de asignación

- La acción de **asignar** `7` a la variable a hará que este nombre o **identificador** se asocie o refiere a la dirección de memoria del objeto `7`, o sea, `a` tendrá la misma identidad que `7`.
- La función `id()` devuelve la **identidad** o lugar de memoria donde se ubica el objeto.

```
>>> id(7)
1449917120
>>> a = 7
>>> id(a)
1449917120
>>> b = 7
>>> id(b)
1449917120
>>> c = a
>>> id(c)
1449917120
```

- Tanto `7`, `a`, `b` o `c` son el mismo objeto en **Python** y ocupan una posición única de memoria

Acción de asignación

- Pero ¿qué pasa si usamos el valor 7.0 en lugar de 7?

```
>>> x = 7.0
>>> id(x)
1722264
>>> type(x)
<class 'float'>
```

- La variable x estará asociada al objeto 7.0, con identidad 1722264, de tipo ***float*** y valor 7.0.
- Los **objetos** de datos simples en Python tienen ***tres*** características:

valor, ***tipo*** e ***identidad***.

- Como se ha notado, las variables no necesitan que sea declarado su tipo antes de ser usadas, como en Pascal o C. Pueden incluso cambiar de tipo a lo largo del programa.

```
>>> x = 7.5
>>> x = 'Hola'
```

Acción de asignación

- Para apreciar mejor el concepto de asignación en programación y diferenciarlo de la simetría del símbolo = en matemáticas, probemos:

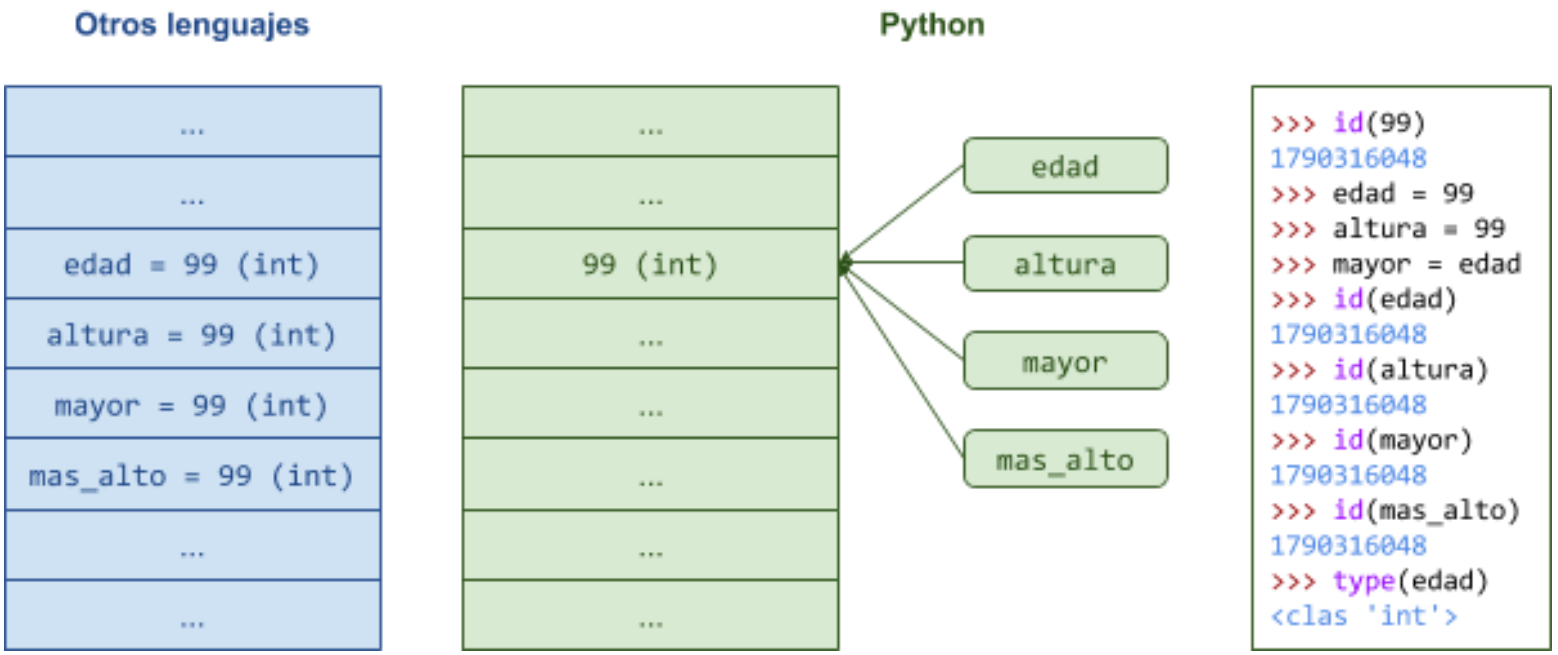
```
>>> 7.0 = x
SyntaxError
>>> x = x + 3
>>> x
10.0
```

- La sentencia `7.0 = x` es un error de **sintaxis** en programación (válido en matemáticas)
- `x = x + 3` sería absurdo en matemáticas, pero en lenguajes de programación significa sumarle a la variable x el valor 3 y el resultado de esa expresión asignarlo luego a la misma variable x.
- Python permite asignaciones múltiples con el uso de tuplas:

```
>>> x, y, z = 7, 8.2, 9
```

Acción de asignación (resumen)

- **En Python una variable** **No** se guarda directamente en la memoria
- Se crea un objeto (cuya posición en la memoria se llama **identidad**)
- Se asocia el **identificador** a la identidad de ese objeto



Identificadores

- Al nombre de una variable lo llamaremos **identificador (*identifier*)**.
- Pero los nombres de otros elementos de los programas, como funciones, clases, librerías también tendrán **identificadores**.
- Los identificadores en programación pueden contener letras y números pero deben empezar siempre con una letra o el carácter guion bajo o subrayado "_".
- Aunque los matemáticos suelen usar nombres con una sola letra para las variables, en programación muchas veces ***es preferible utilizar identificadores con nombres que se asocien con su significado***. Como: area, volumen, lead_III, lado2.
- Hay que hacer notar también que en la mayoría de los lenguajes, incluyendo Python, los identificadores son sensibles al tipo de letra minúscula-mayúscula* . Es decir n y N son variables diferentes.

- ***Ejemplos de identificadores válidos***: *n, pepe, p125, max_entero, A, area*
- ***No Válidos:*** *4abc, dia del mes, area-altura, c+d*

** Python permite identificadores con vocales acentuadas, como área. Aunque no es recomendable esta práctica por si se cambia de lenguaje de programación.*

Identificadores y palabras reservadas

- En los lenguajes de programación hay un grupo de palabras reservadas de operaciones, instrucciones y funciones internas que no pueden usarse como identificadores.
- Se puede imprimir en el Shell de Python la lista de palabras reservadas en este lenguaje, llamadas “keywords”:

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

- Hay que tener cuidado también con las funciones internas o predefinidas del lenguaje (built-in functions) https://docs.python.org/3.3/library/functions.html

Expresiones

- Las expresiones son el mecanismo para hacer cálculos.
- Expresiones se componen de combinaciones de valores e identificadores con operadores.
- Expresiones combinan:
  - Variables
  - Valores
  - Operadores
  - Paréntesis
  - Nombres de funciones especiales (raíz cuadrada, logaritmo, etc)
- Toda expresión tiene un valor que es el resultado de evaluarla de izquierda a derecha, tomando en cuenta las precedencias

```
>>> 1.5*3/2
2.25
>>> 1.2*x + 3  # El valor de x del ejemplo anterior es 10.0
15.0
>>> 3 > (3.1 + 2)/3
True
```

Sentencias o instrucciones

- Las **sentencias** o **instrucciones** son las unidades básicas de los programas (llamados también en el argot de los programadores, **códigos**) que produce una acción, como asignar un valor a una variable, mostrar un resultado, etc.
- El **intérprete** de Python ejecuta cada **sentencia** produciendo la acción dada:

```
>>> y = x/2 + 3   # el valor de x es 10.0
>>> print(y)
8.0
```

- Observar en el siguiente ejemplo que cuando la expresión no se asigna a alguna variable, Python la asigna a una variable llamada _ (guión bajo)

```
>>> 1.5*3/2
2.25
>>> print(_)
2.25
```

Operadores

- Los operadores son los **símbolos** que representan las acciones de cálculo.
- Los operadores los podemos clasificar de 3 tipos:
  - operadores ***aritméticos***,
  - operadores ***lógicos*** o ***booleanos***
  - operadores ***relacionales***.

Operadores Aritméticos: Pueden ser utilizados con datos Reales o Enteros

Operación	Operador	Expresión	Resultado tipo
Suma	+	a + b	Entero si a y b enteros; real si alguno es real
Resta	–	a – b	Entero si a y b enteros; real si alguno es real
Multiplicación	*	a * b	Entero si a y b enteros; real si alguno es real
División, a÷b (real)	/	a / b	Siempre es real
División (entera)	//	a // b	Devuelve la parte entera del cociente a÷b
Módulo (resto)	%	a % b	Devuelve el resto de la división a÷b
Exponenciación, a^b	**	a**b	Entero si a y b enteros; real si alguno es real

Operadores Aritméticos

- Se pueden incluir entre los operadores aritméticos los que operan sobre un solo operando, llamados **unarios**:
  - operador cambio de signo –
  - operador identidad +.
- Ejemplo, -4, +4, --4 equivale a 4

```
>>> 14/4     # División real
3.5
>>> 14//4    # División, devuelve parte entera de dividir 14 entre 4
3
>>> 14%4     # Módulo, devuelve el resto de dividir 14 entre 4
2
```

Operadores Aritméticos con asignaciones

- Llamados también **Operadores de asignación aumentada**
- La acción de incrementar el valor de una variable es muy común en programas informáticos. Por ejemplo, en un contador de eventos, el contador `c` se incrementa en `1`, o `d` se decrementa en 1

```
>>> c = c + 1     # la variable c se incrementa en 1
>>> d = d - 1     # la variable d se decrementa en 1
```

- Python incluye sentencias que compactan el operador de asignación con cualquier operador aritmético. En los casos de incremento o decremento de una variable tenemos:

```
>>> c += 1        # equivale a: c = c + 1
>>> x += 0.01     # equivale a: x = x + 0.01
>>> d -= 2        # equivale a: d = d - 2
```

Operadores Aritméticos con asignaciones

Operación	Operador	Expresión	Equivalente a
Suma y asigna	+=	a += b	a = a+b
Resta y asigna	-=	a -= b	a = a-b
Multiplica y asigna	*=	a *= b	a = a*b
Divide y asigna	/=	a /= b	a = a/b
Divide y asigna la parte entera	//=	a //= b	a = a//b
Módulo y asigna	%=	a %= b	a = a%b
Potencia y asigna	**=	a **= b	a = a**b

Operadores lógicos (Booleanos)

- Estas operaciones son 3: “y lógico” (**and**) o conjunción, “o lógico” (**or**) o disyunción, sobre dos operandos; y la “negación” (**not**) que es un operador unario.
- Los valores lógicos en Python son **True** y **False** para los valores cierto (1 lógico) y falso (0 lógico), respectivamente.

A	B	not A	A or B	A and B
False	False	True	False	False
False	True	True	True	False
True	False	False	True	False
True	True	False	True	True

equivale

A	B	not A	A or B	A and B
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

```
>>> A = True
>>> type(A)
<class 'bool'>
>>> B = False
>>> A or B
True
```

Operadores lógicos. Leyes lógicas

- Algunas leyes lógicas o booleanas. Se pueden demostrar usando la tabla de verdad de cada lado de la igualdad

```
not (not A)  ==   A
A and True   ==   A
A and False  ==   False
A or False   ==   A
A or True    ==   True
```

```
not (A and B) == (not A) or (not B)
not (A or B)  == (not A) and (not B)
```

Leyes de Morgan

Operadores relacionales (*comparison*)

- Son usados para comparar 2 operandos y el resultado es Booleano

– Los operadores relacionales en **Pascal, Delphi** son: =, <>, <, >, <=, >=
– Operadores en lenguajes C, Java, **Python** : ==, !=, <, >, <=, >=
– Operadores en lenguaje **Matlab** : ==, ~=, <, >, <=, >=

Operadores Relacionales				
Matemáticas	En Python	Significado	Ejemplo	Resultado
=	==	Igual a	'a' == 'b'	False
≠	!=	Distinto a	'b' != 'B'	True
<	<	Menor que	7 < 3	False
>	>	Mayor que	7 > 3	True
≤	<=	Menor o igual que	7 <= 7	True
≥	>=	Mayor o igual que	7 >= 3	True

Operadores relacionales (*comparison*)

- Ejemplo de operadores relacionales para chequear que la temperatura medida, que le asignaremos a la variable `temp`, está o no entre 37 y 42 ˚C, ambas inclusive:

```
>>> temp = 38     # temperatura medida
>>> (temp >= 37) and (temp <= 42)
True
>>> 37 <= temp <= 42   # especial de Python
```

- Ejercicio de una expresión que sea cierta cuando, dada una variable `car`, ésta sea un símbolo del alfabeto, o falsa cunado no sea:

```
>>> car = 'q'
>>> (car >= 'a') and (car <= 'z') or (car >= 'A') and (car <= 'Z')
True
>>> # Equivale a
>>> 'a' <= car <= 'z' or 'A' <= car <= 'Z'
True
>>> car = '&'
>>> 'a' <= car <= 'z' or 'A' <= car <= 'Z'
False
```

Orden de las operaciones (precedencia)

- Para resolver expresiones con múltiples operadores, incluyendo diferente tipo de operador, aritmético, booleano o relacional, se debe seguir un **orden de prioridad** para realizar primero una operación y luego otra.
- En caso de dudas, o para mejorar la legibilidad del programa, es recomendable que se usen paréntesis.
- El orden de prioridad o precedencias depende el tipo de lenguaje de programación. En Python las precedencias son similares a C++ y Matlab.
- Máxima prioridad la tiene la realización de las expresiones entre **paréntesis**. Luego la **exponenciación** y las operaciones **unarias**.
- Después siguen las operaciones **aritméticas**, donde se priorizan la multiplicación y división sobre la suma y resta: PEMDAS (Paréntesis, Exponenciación, Multiplicación - División, Adición - Sustracción).
- Después siguen las operaciones **relacionales** y por último las **booleanas**, pero con mayor prioridad de la negación **not**, luego **and** y por último **or**.

Orden de las operaciones (precedencia)

Grupo de operadores	Operador	Precedencia
Parentización	`()`	Mayor
Exponenciación	`**`	
Cambio de signo	`+x, -x`	
Multiplicativos	`*, /, //, %`	
Aditivos	`+, -`	
Comparativos	`==, !=, <, <=, >, >=, in, not in`	
Negación	`not`	
Conjunción	`and`	
Disyunción	`or`	Menor

Orden de las operaciones (precedencia)

- Los operadores exponenciación y unarios (identidad, cambio de signo y negación lógica) **se asocian con el operando de la derecha**.
- El resto de operadores se asocian primero con el **operando de la izquierda**.
- En las expresiones con operadores de igual precedencia se ejecutan de **izquierda a derecha**, exceptuando la exponenciación que va de derecha a izquierda.
- Por ejemplo, la expresión 7/2*3 calcula primero la división 7 entre 2 y el resultado los multiplica por 3, resultando 10.5.
- Si se quiere dividir entre el producto 2*3, hay que encerrarlos entra paréntesis: 7/(2*3).
- La asociación por la **derecha** de la exponenciación se puede ver en la siguiente expresión 2**4**2 y (2**4)**2.

```
>>> 2**4**2
65536
>>> (2**4)**2
256
```

Ejemplos de expresiones y precedencias (Python).

- 24 / 5**2 - 0.96 equivale a (24 /(5**2)) – 0.96 es igual a 0.0
- 2 + 7 // 3 * 2 -15 → 2 + 2 * 2 – 15 → 2 + 4 – 15 → 6 – 15 → -9
- 3 * 5 % 4 + 11 // 4 * 3 → 15%4+11//4*3 → 3+2*3 → 9
- True and not False or False → True
- 1 >= 7 % 2 → 1 >= 1 → True
- 1 % 2 > 0.5 → 1 > 0.5 → True
- True or not and (2 > 1) Sintaxis incorrecta (not and)
- (3 // 2) > (3 % 2) and not True → False and False → False
- (3 // 2 > 3 % 2) and not True → (1 > 1) and False → False and False → False
- *Escribir expresión equivalente a:* **a no está entre –1 y 1 ni entre 2 y 3**

  not (a>= -1 and a <=1) and not (a >= 2 and a <= 3)
- *Escribir expresión:* ***a*** **es un múltiplo de 10 ubicado entre 3000 y 4000**

  a % 10 == 0 and (a>= 3000 and a <= 4000)

Operaciones con texto (strings)

- Los ***strings*** no pueden operarse matemáticamente.
- Sin embargo el operador **+** sí que realiza la interesante acción de **concatenar** dos strings.
- También el operador ***** repite el string un número de veces dado
- Operadores lógicos en strings: **in** , **not in**

```
>>> '40' + '8'
'408'
>>> Name = 'Luis'
>>> Apellido = 'Garcia'
>>> Name + ' ' + Apellido
'Luis Garcia'
>>> 'Ven'*4          # equivale a 'Ven'+'Ven'+'Ven'+'Ven'
'VenVenVenVen'
>>> 10*'-'
'----------'
>>> 'h' in 'hola'
True
>>> 'H' not in 'hola'
True
```

Lectura de datos

- En Python se utiliza la función interna **input()**.
- Se utiliza un mensaje de texto para indicar al usuario del programa qué tiene que introducir.
- Lo que se teclea y se introduce al programa es un valor de tipo string.
- Por lo que hay que tener en cuenta que si los datos son numéricos habrá que convertirlos de **string** a enteros (`int`) o a reales (`float`).

```
>>> Nombre = input('Cómo te llamas? ')
Cómo te llamas? José
>>> type(Nombre)
<class 'str'>
>>> Edad = int(input('Introduce tu edad: '))
Introduce tu edad: 21
>>> type(Edad)
<class 'int'>
>>> Altura = float(input('Cuánto mides? '))
Cuánto mides? 1.78
>>> type(Altura)
<class 'float'>
```

Conversión entre tipos de datos

- Funciones internas **float**(), **int**(), **str**(). Además, **bin**() -> integer - binario

```
>>> float('123')
123.0
>>> float(Edad)  # Variable Edad de tipo int del ejemplo anterior
21.0
>>> float('abc')
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    float('abc')
ValueError: could not convert string to float: 'abc'
```

```
>>> int('123')
123
>>> int(27.8)
27
>>> int(-24.9)
-24
```

```
>>> str(254)
'254'
>>> str(1/3)
'0.3333333333333333‘
>>> bin(255)
'0b11111111'
>>> bin(256)
'0b100000000'
```

Escritura de datos

- En Python se utiliza la función interna **print()**.
- La función **print()** escribe como string todo su argumento (entrada) como una secuencia de expresiones separadas por comas o concatenadas.

```
>>> a = 3.5
>>> b = 15
>>> print('El producto de', a ,  'por', b ,  'es', a*b)
El producto de 3.5 por 15 es 52.5
>>> print('El producto de '+str(a)+' por '+str(b)+' es '+str(a*b))
El producto de 3.5 por 15 es 52.5
```

- Palabras reservadas del argumento de la función **print**() que indican:
  - cómo terminar la escritura (**`end`**)
  - y cómo separar (**`sep`**) los elementos.
- Cuando se quiera terminar la función **print**() sin que salte a una nueva línea, se agrega al final del argumento **`end`**`=''`
- Cuando no se usa el **`end`**, el valor por omisión es salto a nueva línea, que sería **`end`**`='\n'`.
- En lugar de separar las secuencias por espacios en blanco, se puede omitir la separación (**`sep`**`=''`)

Escritura de datos

File Edit Format Run Options Window Help

```
print('Q', end='')
print('u', end='')
print('é!', end='\n') # equivale a print('é!')
print('A', 'B', 'C')
print('A', 'B', 'C', sep=',')
print('A', 'B', 'C', sep='')
print('10','27','59', sep=':')
print('10','27','59', sep='----')
```

```
Qué!
A B C
A,B,C
ABC
10:27:59
10----27----59
>>>
```

Comentarios

- Es una buena opción comentar en lenguaje natural lo que las instrucciones del programa están haciendo

```
"""
Created on Mon Jan  4 23:59:52 2016

@author: gomis
"""

# Cálculo de la frecuencia cardíaca
RR = 0.875      # Intervalo R-R en segundos
freq = 1/RR*60  # Frecuencia en pulsaciones por minuto
```

Errores

- Sintácticos

```
>>> 4/*8
SyntaxError: invalid syntax
>>> pirnt('Hola mundo!')
Traceback (most recent call last):
  File "<pyshell#85>", line 1, in <module>
    pirnt('Hola mundo!')
NameError: name 'pirnt' is not defined
```

- De ejecución

```
>>> alumnos = 56
>>> grupos = 0
>>> alum_grupo = alumnos/grupos
Traceback (most recent call last):
  File "<pyshell#90>", line 1, in <module>
    alum_grupo = alumnos/grupos
ZeroDivisionError: division by zero
```

- Semánticos

```
>>> base = 4
>>> altura = 2
>>> area = base / altura
>>> print(area)
2.0
```