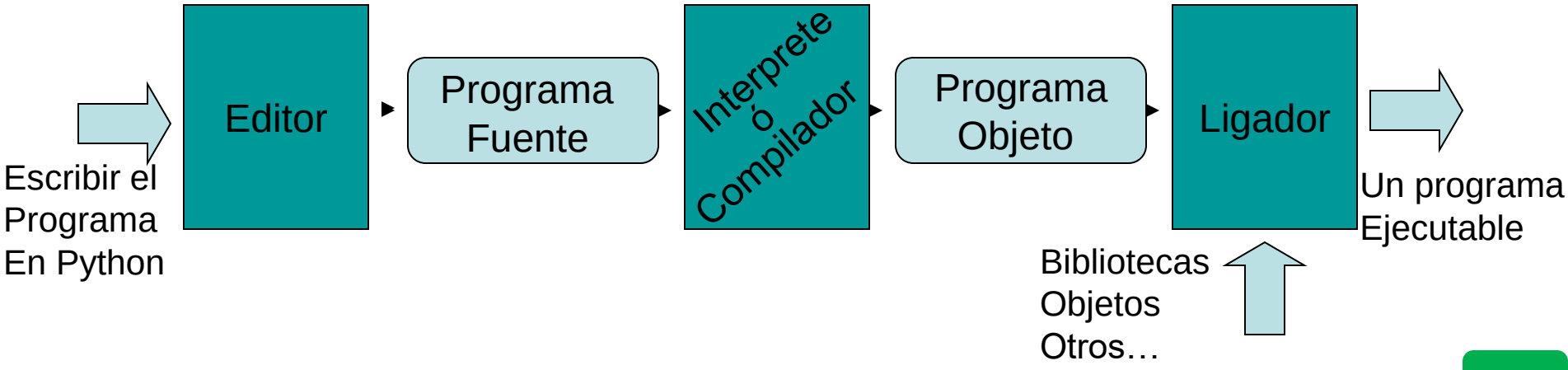
CAPITULO 2: UNA CALCULADORA AVANZADA

Sesiones INTERACTIVAS

- Ambiente de desarrollo. IDE (Integrated Develop. Envirom)

Aplicación de software que provee facilidades al programador para desarrollar software. Compuesto normalmente por un **editor de código fuente, un compilador o interprete, herramientas de automatización, depuradores y analizadores de tiempo real.**



Sesiones INTERACTIVAS

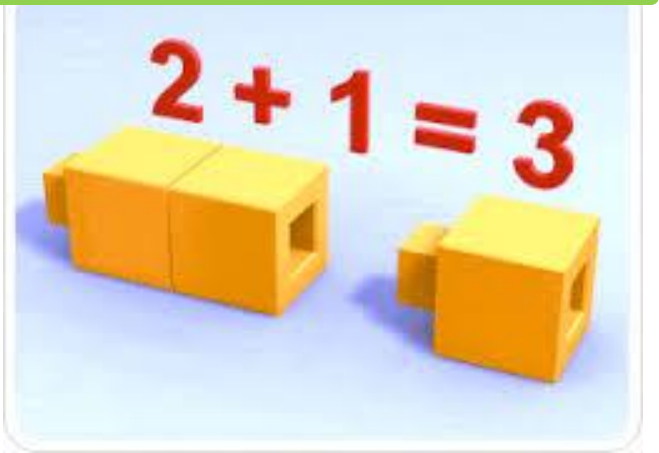
- Lenguajes interpretados, OFRECEN una herramienta INTERACTIVA. Lo que se uso la clase pasada.

- Con esta herramienta **no es necesario escribir un programa completo**. Se pueden dar ordenes individuales.

- Podemos **«dialogar»** con el interprete.

- Es de mucha utilidad para observar el comportamiento de un fragmento de código.

Operadores Aritméticos

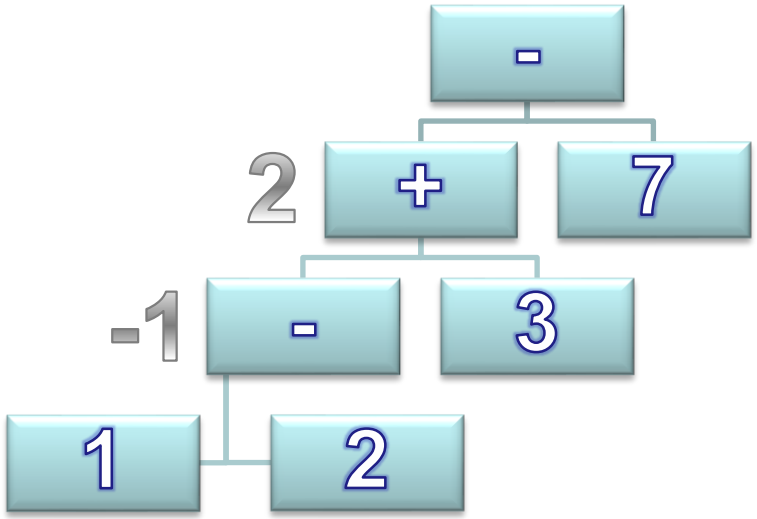


- Operadores de SUMA Y RESTA
- Son operadores **binarios**, operan sobre dos operandos numéricos.
- Se pueden introducir mas de una operación, como se resuelven?...
- Son **asociativos** por izquierda.

Operadores Aritméticos

- Cual será el resultado de la siguiente expresión, hágalo a mano y luego compruébelo en el interprete.

```
>>> 1 – 2 + 3 - 7
```

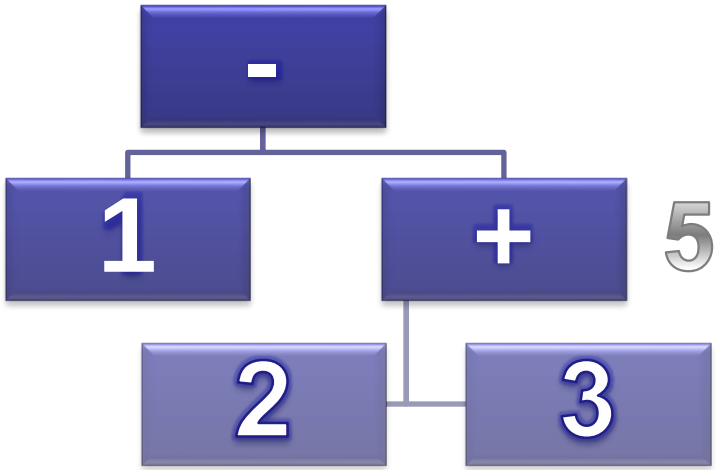


-5

Operadores Aritméticos

- Si hay paréntesis, se evalúan primero!

```
>>> 1 – (2 + 3)
```



5

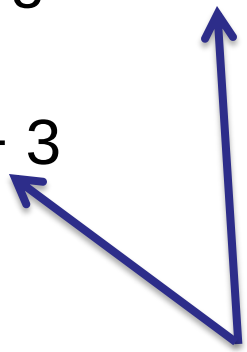
-4

En el arbol, no aparecen paréntesis, su estructura indica prioridad!

Operadores Aritméticos

- Operadores **UNARIOS**

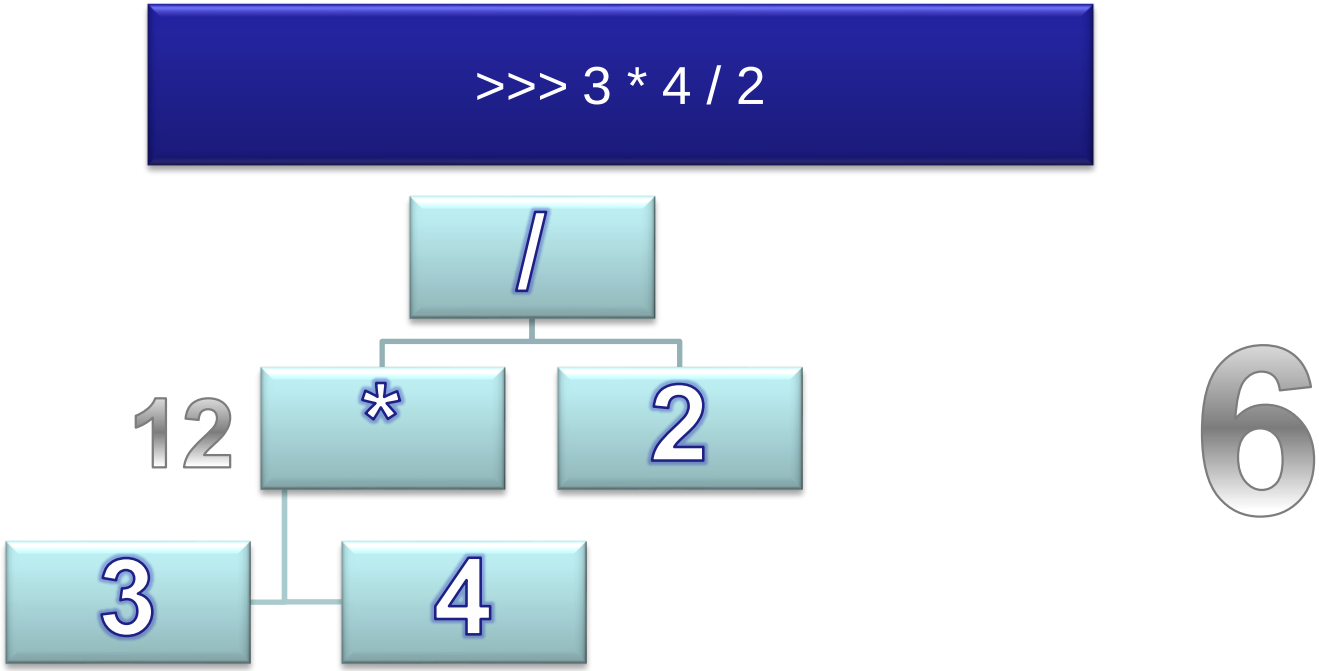
- Operan sobre **solo un operando** numérico.
- Cambio de signo: - (2+3)
- Identidad: + 3



Operadores Aritméticos

- Operadores de **Multiplicación y División**.

- También <u>asociativos por izquierda</u>.

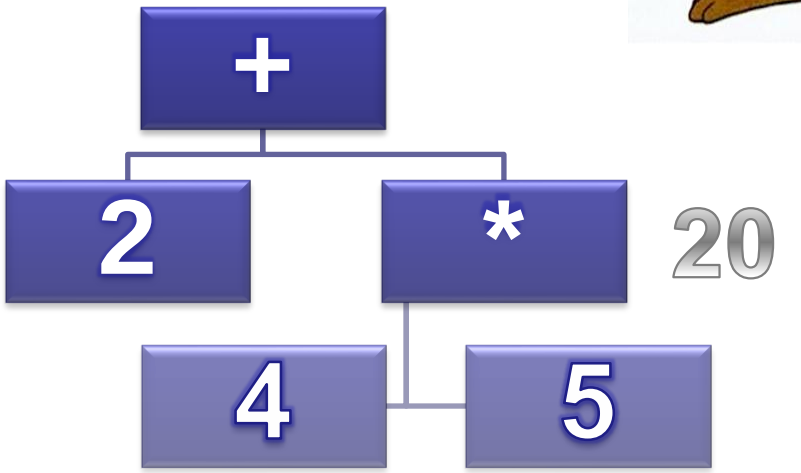


Operadores Aritméticos

- Al combinar operadores aritméticos se debe verificar el ORDEN DE PRECEDENCIA. *** y / tienen mayor precedencia.**

```
>>> 2 +  4 * 5
```

22 ?



Operadores Aritméticos

- Y si quiero **alterar** el orden de precedencia!?....debo utilizar **PARENTESIS**.
- Operador **MODULO**. 
- Igual precedencia que multiplicación y división. Asociativo por izquierda.
- Atención!!!...devuelve el **RESTO** de la división entera entre dos OPERANDOS!..

Operadores Aritméticos

- Ejemplos:

```
>>> 27 %  5
    2

>>>25  %  5
    0
```

%

Y ahora:

3 % 16?

Operadores Aritméticos

**

- Operador **EXPONIENCIACION**.
- **Mayor precedencia** que los otros operadores. Asociativo por **DERECHA!**.
- 2^{3^2}

```
>>> 2 ** (3 ** 2)
>>> 2 **  (9)
>>> 512
```

Operadores Aritméticos

- Un ejemplo:

- **Mayor precedencia** que los otros operadores. Asociativo por **DERECHA!**.

```
>>>2 + 3 ** 2 * 5
```

```
>>> 2 + (3 ** 2) * 5
>>> 2 +  (9) * 5
>>> 47
```

Operadores Aritméticos

- Tabla de operadores ARITMETICOS

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4

Tabla 2.1: Operadores para expresiones aritméticas. El nivel de precedencia 1 es el de mayor prioridad y el 4 el de menor.

Operadores Aritméticos

- De acuerdo a la tabla de precedencia vista:

- Cual seria el resultado esperado de ejecutar en python la siguiente expresión aritmética:

```
-3**2
```

Comprende el porque?

Errores y excepciones

- Al introducir una expresión es factible **equivocarnos**.

- El interprete nos lo indica con un **mensaje de error**

- Este mensaje contiene información sobre el **tipo** de error y el **lugar** donde se encuentra.

- En python los errores se denominan «**excepciones**».

Errores y excepciones

```
>>> 1 + * 3↲
  File "<stdin>", line 1
    1 + * 3
        ^
SyntaxError: invalid syntax
>>> 2 + 3 %↲
  File "<stdin>", line 1
    2 + 3 %
          ^
SyntaxError: invalid syntax
>>> 1 / 0↲
Traceback (innermost last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo
```

Indica posición del error

Error de sintaxis

No hay error de sintaxis.
Error de división por 0.

Tipos de Datos

- Cada valor utilizado por python pertenece a un tipo de datos.

- Al realizar una operación se tiene en cuenta el tipo de datos de la operación y el resultado será del mismo tipo.


```
>>> 3 / 2
1
```

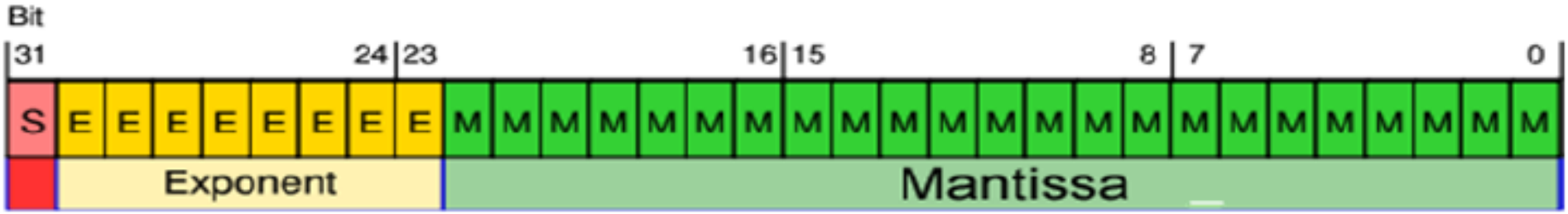
```
>>> 3.0 / 2.0
1.5
```

Tipos de Datos

- Enteros y Flotantes.
- Los números con decimales, se suelen llamar en matemáticas números **reales**.
- En python, usamos meras aproximaciones a los números reales, ya que la capacidad es limitada, y no infinita.
- Esta aproximación es un standard IEEE 754.
- **Se los denomina números de coma flotante, o flotantes.**

Tipos de Datos

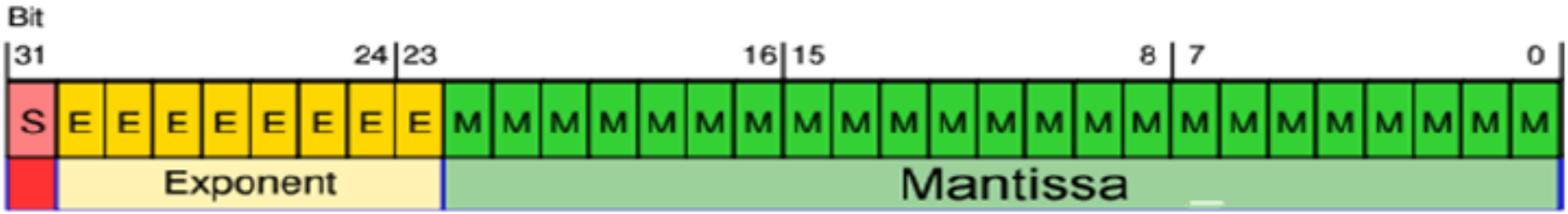
- Un flotante debe cumplir especificaciones según IEEE754



Tipos de Datos

- Un flotante consta de 3 partes.
- El signo.
- El exponente.
- Y la mantisa

- El exponente se separa de la mantisa con la letra e E.
- **2e3** equivale a 2 de exponente y 3 de mantisa osea 2×10^3

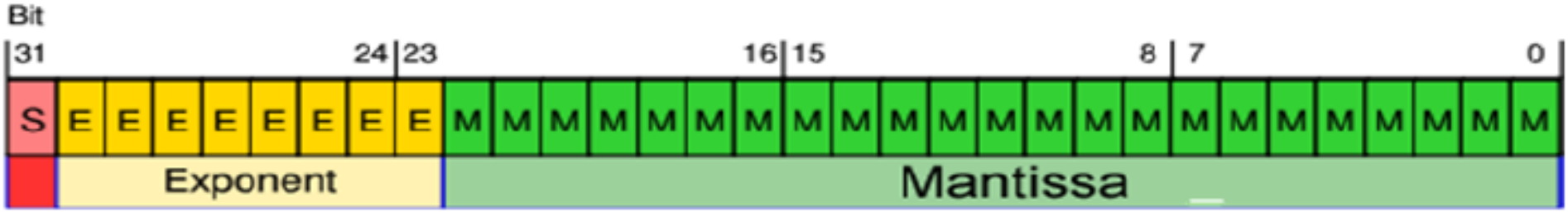


Tipos de Datos

- El exponente puede ser negativo.
- 2e-3 equivale a 2.1 x 10^{-3} que es 0.0021

- Si la parte entera de un flotante es nula puede empezar con «.», el 0 es opcional. **Por lo tanto 0.3 equivale a .3**

- A su vez, si la parte decimal es cero, en un flotante el punto es obligatorio. **Por lo tanto 2.0 equivale a 2.**



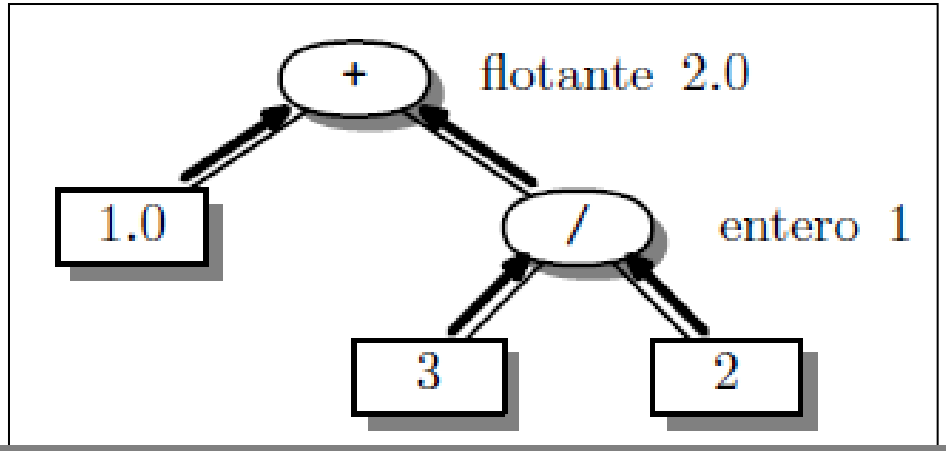
- En una operación, si los operandos son de distinto tipo, el resultado es del **tipo más general**.

- Un flotante es un tipo de dato **más general** que un entero.

```
>>>1.0 + 3 / 2
```

2.0

- En este caso, la regla no se cumple?



Valores Lógicos

- Desde la versión 2.3 de Python tenemos un tipo de dato que permite expresar solo dos condiciones.

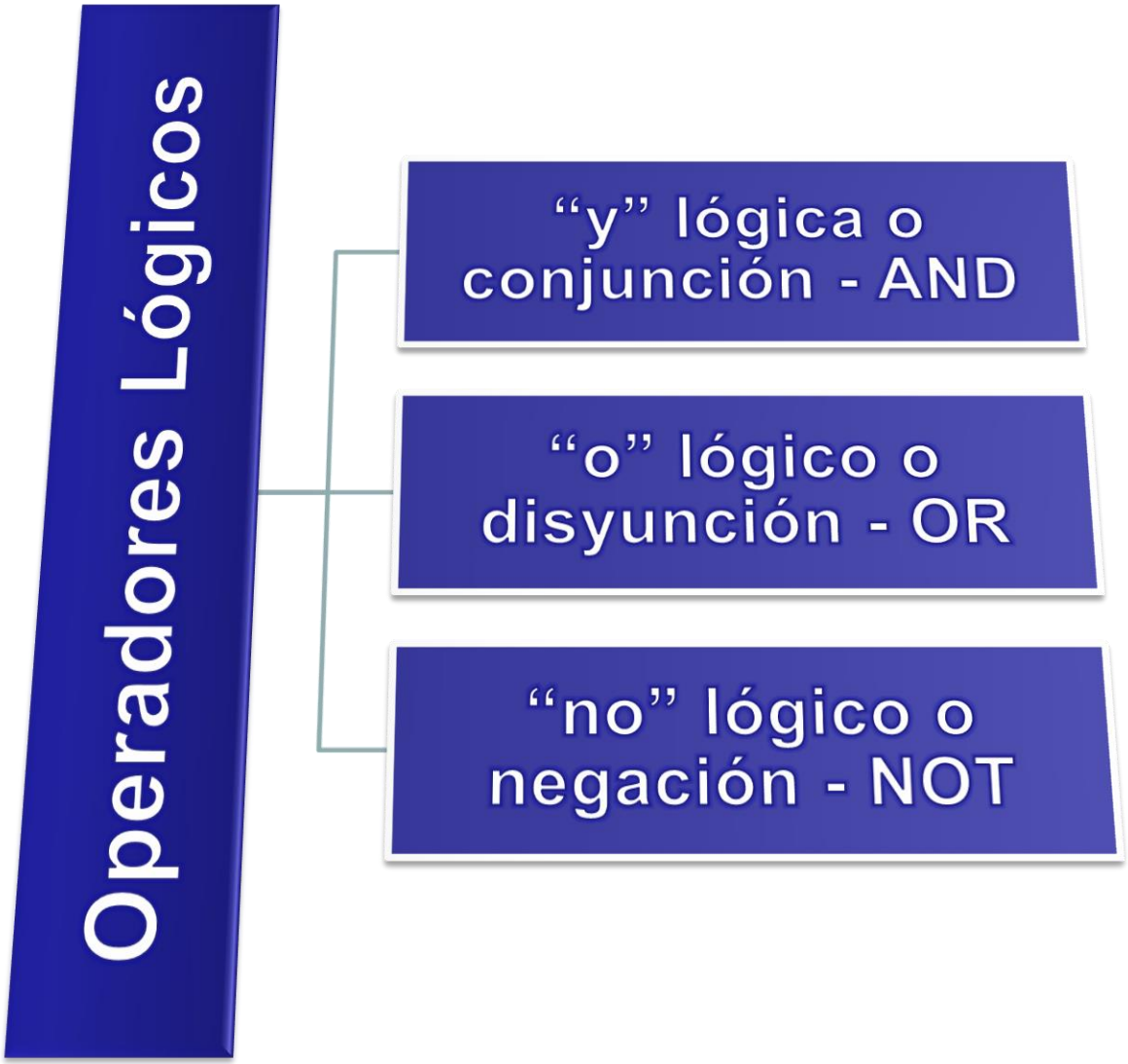
CIERTO FALSO

- El valor cierto se expresa con «True»
- El valor falso se expresa con «False»

- Estos se denominan **valores lógicos** o booleanos.

- Existen operadores especiales, llamados **OPERADORES LOGICOS**, para este tipo de datos.

Operadores Lógicos



Operadores Lógicos

- **«Y» lógica:**

- Su resultado es TRUE, solo si ambos operandos lo son.

and		
operandos		resultado
izquierdo	derecho	
True	*True*	*True*
True	*False*	*False*
False	*True*	*False*
False	*False*	*False*

Operadores Lógicos

- **«O» lógico:**

- Su resultado es TRUE, si alguno o ambos operandos son TRUE.
- Su resultado es FALSE solo si ambos operandos lo son.

or		
operandos		resultado
izquierdo	derecho	
True	*True*	*True*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

Operadores Lógicos

- **«NO» lógico:**

- Este operador es UNARIO. Proporciona valor TRUE si el operando es FALSE y viceversa.

not	
operando	resultado
True	*False*
False	*True*

Operadores Lógicos

- Los operadores lógicos, pueden combinarse y formar **expresiones lógicas.**

- Para esto se debe considerar el orden de precedencia.

Operación	Operador	Aridad	Asociatividad	Precedencia
Negación	**not**	Unario	—	alta
Conjunción	**and**	Binario	Por la izquierda	media
Disyunción	**or**	Binario	Por la izquierda	baja

True and False or True ?....

Operadores de Comparación

- Este conjunto de operadores devuelve resultados booleanos.

- El operador «**igualdad**» devuelve True si los valores comparados son iguales.

```
>>> 2 == 3
False
>>> 2 == 2
True
>>> 2.1 == 2.1
True
>>> True == True
True
>>> True == False
False
>>> 2 == 1+1
True
```

Operadores de Comparación

- Otros operadores de comparación:

operador	comparación
==	es igual que
!=	es distinto de
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

Tabla 2.3: Operadores de comparación.

Operadores de Comparación

- Todos los operadores vistos pueden utilizarse en combinación, generando la siguiente tabla de precedencia:

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4
Igual que	==	Binario	—	5
Distinto de	!=	Binario	—	5
Menor que	<	Binario	—	5
Menor o igual que	<=	Binario	—	5
Mayor que	>	Binario	—	5
Mayor o Igual que	>=	Binario	—	5
Negación	not	Unario	—	6
Conjunción	and	Binario	Por la izquierda	7
Disyunción	or	Binario	Por la izquierda	8

Operadores de Comparación

- Caso especial, asociatividad de comparadores:

```
2 < 3 < 4
```

- Si fuera asociativo por izquierda, 2<3? Seria true….
- Y luego true<4???....NOOOOO!
- Cuando aparece una sucesión de comparadores Python lo resuelve:

```
(2 < 3) and (3 < 4)
```

Variables y asignaciones

- En la programación, a veces es necesario recordar un valor determinado para utilizarlo posteriormente y repetirlo sin cometer errores.

- Para esto pueden utilizarse **VARIABLES**.

- Al colocar un nombre, y luego el símbolo «=» estamos creando una variable y se le asigna un valor.

```
>>> pi=3.14159265358
```

- El acto de dar valor a una variable se denomina **asignación**.

Variables y asignaciones

- Luego de esta instrucción, escribir pi, o el numero completo es IDENTICO.

- En el interprete, asignar valores a variables, **NO GENERA UNA SALIDA POR PANTALLA.**

- Si queremos saber el valor de la variable, podemos colocar su nombre…

```
Python 2.7.3 (default, Apr 10 2012,
Type "copyright", "credits" or "lic
[Pulsa Control-d para reiniciar el
>>> pi=3.141516
>>>
```

```
Python 2.7.3 (default, Apr 10 2
Type "copyright", "credits" or
[Pulsa Control-d para reiniciar
>>> pi=3.141516
>>> pi
3.141516
>>>
```

Variables y asignaciones

- Una asignación tiene la forma:

Variable = expresión

- Primero se **evalúa** la expresión a la derecha del símbolo =.
- Luego se **asigna** a la variable.
- Se puede asignar valor reiteradas veces a la misma variable, esta solo recordara el **ultimo valor asignado**.

Variables y asignaciones

- Una asignación no es una ecuación:

```
>>> x = 3 ↲
>>> x = x + 1 ↲
>>> x ↲
4
```

- **<u>Primero se evalúa la expresión a la derecha del símbolo =.</u>**

- Luego se asigna a la variable.

Variables y asignaciones

- El nombre de una variable es su identificador. Hay reglas para crear los mismos.

- Un identificador puede estar formado por:
  - Letras minúsculas
  - Letras mayúsculas
  - Dígitos
  - Carácter subrayado

- **No puede comenzar con un digito**.

- No puede coincidir con una palabra reservada o clave.

Variables y asignaciones

- Python distingue entre MAYUSCULAS y minúsculas. Por lo que no son las mismas variables

PI pi

- **NO PUEDE UTILIZARSE EL ESPACIO EN BLANCO EN UN IDENTIFICADOR**

- **Siempre tratar que el identificador GUARDE UNA RELACION con la variable que representa.**

Asignaciones con operador

- La siguiente expresión es muy utilizada. Es un incremento en 1 en el valor de la variable. Un acumulador.

```
i = i + 1
```

- Y al ser tan utilizado puede escribirse de la siguiente forma:

```
i += 1
```

- OJO – Sin espacios entre los símbolos ‘+’ y ‘=’

Asignaciones con operador

- Esto puede realizarse con todos los operadores!.

```
z += 2
z *= 2
z /= 2
z -= 2
z %= 2
z **= 2
```

- Cual será el resultado de:

-64

```
>>> z = 2
>>> z += 2
>>> z += 2 - 2
>>> z *= 2
>>> z *= 1 + 1
>>> z /= 2
>>> z %= 3
>>> z /= 3 - 1
>>> z -= 2 + 1
>>> z -= 2
>>> z **= 3
>>> z
```

Variables NO INICIALIZADAS

- **En Python, la primera operación sobre una variable DEBE ser la ASIGNACION de un valor.**

- No puede usarse una variable a la cual no se le asigno un valor. Esto provoca un error de NOMBRE.

```
>>> a + 2↵
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'a' is not defined
```

- La asignación de un valor es inicializar la variable.

Tipo de datos: CADENAS

- Hasta ahora hemos visto dos tipos de **datos numéricos**. Flotantes y enteros.
- Existe un tipo de datos, que almacena información textual. **Estos son CADENAS**.
- Son una secuencia de caracteres (letras, números, espacios, comas, puntos, guiones, etc) ENCERRADAS entre comillas simples o dobles.
- Ej: ‘un ejemplo’ ‘otro ejemplo’ ….

- Las cadenas también pueden ALMACENARSE EN VARIABLES.

```
>>> nombre = 'Pepe' ↲
>>> nombre ↲
'Pepe'
```

- Se puede operar con cadenas, la suma equivale a unir.

```
>>> 'a' + 'b' ↲
'ab'
>>> nombre = 'Pepe' ↲
>>> nombre + 'Cano' ↲
'PepeCano'
>>> nombre + '␣' + 'Cano' ↲
'Pepe Cano'
>>> apellido = 'Cano' ↲
>>> nombre + '␣' + apellido ↲
'Pepe Cano'
```

Tipo de datos: CADENAS

- Esta unión de cadenas se llama **CONCATENACION**.

- OJO! no es lo mismo sumar que concatenar.

```
>>> '12' + '12'↲
'1212'
>>> 12 + 12↲
24
```

Tipo de datos: CADENAS

- Además existe un operador de **repetición** de cadenas.
- El mismo utiliza el símbolo ‘*’ .
- Y opera sobre dos valores, una cadena y un dato entero.
- El resultado es otra cadena, que se repite tantas veces como lo indicaba el numero entero.

```
>>> 'Hola' * 5↵
'HolaHolaHolaHolaHola'
```

Tipo de datos: CADENAS

- Se puede comparar CADENAS?....SI
- Que sean iguales, implica ser iguales carácter a carácter.
- Pero MAYORES O MENORES?
- Orden Alfabético. Es menor si la precede en un diccionario.
- Se utiliza el código ASCII.
- Puede usarse función **ord(‘a’)** para saber el valor numérico del mismo. Y **chr(’97’)** para hacer la inversa.

Tipo de datos: CADENAS

- Mayúsculas valor numérico INFERIOR.
- Letras acentuadas, son siempre mayores.

FUNCIONES predefinidas

- Además de los operadores aritméticos básicos. Python proporciona funciones predefinidas.

- Por ej. la **función ABS**, proporciona el valor absoluto del numero entre paréntesis.

- El numero sobre el que se aplica la función es ARGUMENTO

```
>>> abs(-3) ↲
3
>>> abs(3) ↲
3
```

FUNCIONES predefinidas

- Existen otras funciones como:
- **Función: FLOAT (argumento)**
- Esta función, convierte en FLOTANTE el argumento dado.
- El argumento puede ser una cadena de números, sino proporcionara error.

```
>>> float('3.2')
3.2
>>> float('3.2e10')
32000000000.0
```

FUNCIONES predefinidas

- **Función: INT (argumento)**
- Esta función, convierte en INT el argumento dado.
- Un argumento FLOAT ***es eliminada su parte FRACCIONARIA.***
- Funciona de manera análoga con FLOAT.

```
>>> int(2.1) ↲
2
>>> int(-2.9) ↲
-2
```

FUNCIONES predefinidas

- **Función: STR (argumento)**
- Esta función, convierte en STRING, CADENA, un argumento dado.
- **Función: ROUND(argumento).**
- Esta función puede usarse con uno o dos argumentos.
- Con un argumento redondea al flotante mas cercano con parte fraccionaria nula.
- El resultado siempre es de **tipo flotante**

FUNCIONES predefinidas

- **Función: ROUND(arg1, arg2)**
- El segundo argumento indica el numero de decimales que deseamos conservar tras el redondeo.

```
>>> round(2.1451, 2)
2.15
>>> round(2.1451, 3)
2.145
>>> round(2.1451, 0)
2.0
```

- Ojo, para un comportamiento homogéneo, usar funciones floor, y ceil. Se explicaran mas adelante.

- **Estas funciones pueden combinarse en expresiones:**

- **Intente comprender los siguientes ejemplos:**

abs(-23) % int (7.3)	2
abs(round(-34.2765,1))	34.3
str(float(str(2) *3 + '.123')) + '321'	222.123321

FUNCIONES definidas MODULOS

- **Existen otras funciones no disponibles al iniciar sesión con Python.**
- **Debemos indicarle al lenguaje que las incorpore por que las utilizaremos.**
- **Estas funciones están definidas en MODULOS aparte.**
- **El modulo MATH. Contiene funciones como seno, coseno, etc.**

FUNCIONES definidas MODULOS

- **Para importar una sola función del modulo MATH:**

```
from math import sin
```

- **El argumento de las funciones trigonométricas debe expresarse en RADIANES**

```
from math import cos
```

- **De esta manera se importo el significado de la función coseno.**

FUNCIONES definidas MODULOS

- **En una misma sentencia podemos importar más de una función.**

```
from math import sin, cos
```

- **Puede ser tedioso importar varias funciones, se pueden importar TODAS las funciones del modulo con:**

```
from math import *
```

- **Algunas de las demás funciones del modulo MATH**

sin(*x*)	Seno de *x*, que debe estar expresado en radianes.
cos(*x*)	Coseno de *x*, que debe estar expresado en radianes.
tan(*x*)	Tangente de *x*, que debe estar expresado en radianes.
exp(*x*)	El número *e* elevado a *x*.
ceil(*x*)	Redondeo hacia arriba de *x* (en inglés, «ceiling» significa techo).
floor(*x*)	Redondeo hacia abajo de *x* (en inglés, «floor» significa suelo).
log(*x*)	Logaritmo natural (en base *e*) de *x*.
*log*10(*x*)	Logaritmo decimal (en base 10) de *x*.
sqrt(*x*)	Raíz cuadrada de *x* (del inglés «square root»).

FUNCIONES definidas MODULOS

- **Además el modulo contiene el valor de constantes.**

```
>>> pi
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'pi' is not defined
>>> from math import pi,e
>>> pi
3.141592653589793
>>> e
2.718281828459045
```

FUNCIONES definidas MODULOS

- **Existe un sinnúmero de módulos para cada campo de aplicación especifico. Esta es una de las potencialidades de python.**

- **Otro ejemplo MODULO SYS**

- Que contiene funciones como **exit**, que **aborta** inmediatamente la ejecución del interprete.

- Otra es **MAXINT**, que almacena de acuerdo al computador el mayor valor de entero aceptado.

- **Existes MANUALES de REFERENCIA de los módulos.**

Métodos

- **Determinados tipos de datos, permiten invocar funciones especiales.**
- **De los tipos de datos visto, las CADENAS permiten invocar algunas funciones especiales llamadas METODOS.**
- **Un método permite modificar o convertir una cadena.**
- Por ejemplo, el método **.lower** convierte una cadena completa a minúsculas.

Métodos

- La **sintaxis** de los métodos es diferente.
- Primero aparece el **nombre de la cadena** sobre la cual se aplicara el método.
- El nombre del método se separa del objeto con un **PUNTO.**
- Los **paréntesis abierto y cerrado** finales son obligatorios.

```
>>> cadena = 'Un EJEMPLO de Cadena'
>>> cadena.lower()
'un ejemplo de cadena'
>>> 'OTRO EJEMPLO'.lower()
'otro ejemplo'
```

Métodos

- Otro método, .**upper.**
- El cual devuelve toda la cadena en MAYUSCULAS.

- Otro método, .**title**
- El cual pasa a MAYUSCULA la Inicial de cada Palabra.

Métodos

- Hay métodos, que aceptan **PARAMETROS**.
- El método .**REPLACE**, recibe dos cadenas de argumento.
- Busca una de ellas y la reemplaza por la otra todas las veces que la primera aparezca.
- Hay métodos, que tienen funciones equivalentes….usar una u otra es una opción del programador.

Métodos

- Ejemplos:

```
>>> cadena = 'Un EJEMPLO de Cadena'
>>> cadena.lower()
'un ejemplo de cadena'
>>> 'OTRO EJEMPLO'.lower()
'otro ejemplo'
```

```
>>> 'Otro ejemplo'.upper()
'OTRO EJEMPLO'
```

Métodos

- Ejemplos:

```
>>> 'PEDRO␣F.␣MAS'.title() ↲
'Pedro F. Mas'
>>> 'Juan␣CANO'.title() ↲
'Juan Cano'
```

```
>>> 'un␣pequeño␣ejemplo'.replace('pequeño', 'gran') ↲
'un gran ejemplo'
>>> una_cadena = 'abc'.replace('b', '-') ↲
>>> una_cadena ↲
'a-c'
```

REPASEMOS!!!!

- Que aprendimos:

✓ Sesiones, son interactivas. Podemos dialogar con el interprete. (solo pythong)

✓ Operadores ARITMETICOS (+ - * / % **) y su precedencia

✓ Errores y excepciones. Python nos indica con un mensaje tipo de error y ubicación.

✓ Tipos de datos
- ✓ Enteros
- ✓ Flotantes
- ✓ Cadenas

REPASEMOS!!!!

- ✓ Valores LOGICOS – TRUE & FALSE
- ✓ Operadores LOGICOS (and – or – not)
- ✓ Operadores de comparación (== != < <= > >=) y sus precedencias
- ✓ Asignar valores a VARIABLES. Utilizables en los programas y evitan errores.
- ✓ Asignaciones con operador. (+= -= *= /= %= **=)

REPASEMOS!!!!

- ✓ Funciones predefinidas como abs, float, int, str y round.
- ✓ Funciones definidas en MODULOS
- ✓ Modulo de funciones MATH
- ✓ Modulo de funciones SYS

- ✓ Métodos, o funciones especiales para CADENAS.
  - ✓ Método .lower
  - ✓ Método .upper
  - ✓ Método .title
  - ✓ Método .replace

A PRACTICAR!!!!

- Ejercicios para TRABAJAR:

▶ 20 Evalúa el polinomio $x^4 + x^3 + 2x^2 - x$ en $x = 1.1$. Utiliza variables para evitar teclear varias veces el valor de x. (El resultado es 4.1151.)

▶ 21 Evalúa el polinomio $x^4 + x^3 + \frac{1}{2}x^2 - x$ en $x = 10$. Asegúrate de que el resultado sea un número flotante. (El resultado es 11040.0.)

A PRACTICAR!!!!

- Ejercicios para TRABAJAR:

EJERCICIOS

▶ 23 Evalúa estas expresiones y sentencias en el orden indicado:

a) `a = 'b'`

b) `a + 'b'`

c) `a + 'a'`

d) `a * 2 + 'b' * 3`

e) `2 * (a + 'b')`

A PRACTICAR!!!!

- Ejercicios para TRABAJAR:

EJERCICIOS

▶ 26 ¿Qué resultados se muestran al evaluar estas expresiones?

```
>>> 'abalorio' < 'abecedario'↲
>>> 'abecedario' < 'abecedario'↲
>>> 'abecedario' <= 'abecedario'↲
>>> 'Abecedario' < 'abecedario'↲
>>> 'Abecedario' == 'abecedario'↲
>>> 124 < 13↲
>>> '124' < '13'↲
>>> '␣a' < 'a'↲
```

A PRACTICAR!!!!

- Ejercicios para TRABAJAR:

EJERCICIOS

▶ 30 ¿Qué resultados se obtendrán al evaluar las siguientes expresiones Python? Calcula primero a mano el valor resultante de cada expresión y comprueba, con la ayuda del ordenador, si tu resultado es correcto.

a) *int*(*exp*(2 * *log*(3)))

b) *round*(4**sin*(3 * *pi* / 2))

c) *abs*(*log*10(.01) * *sqrt*(25))

d) *round*(3.21123 * *log*10(1000), 3)