

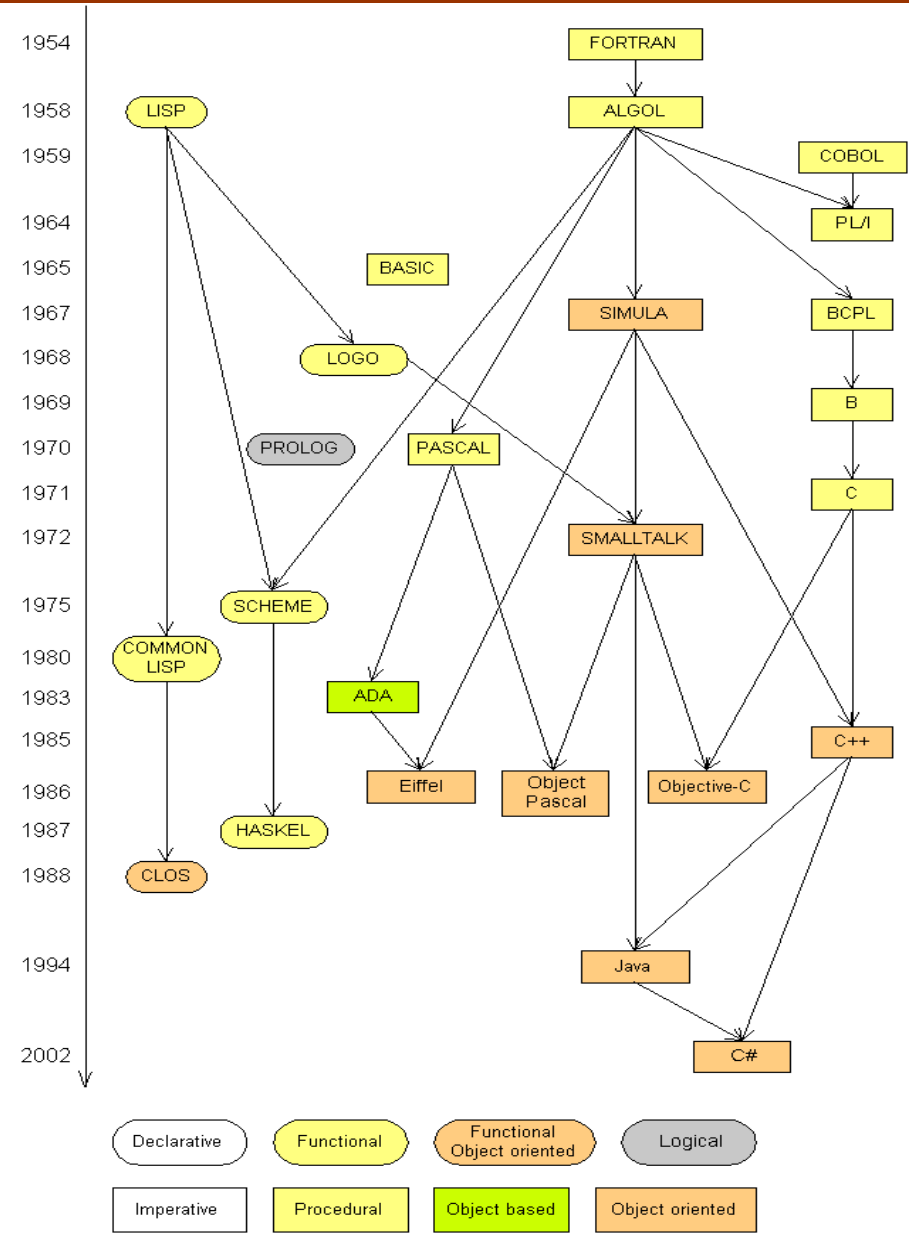
# Programación en Python

**CTIC - UNI**

**Victor Melchor**

# Lenguajes

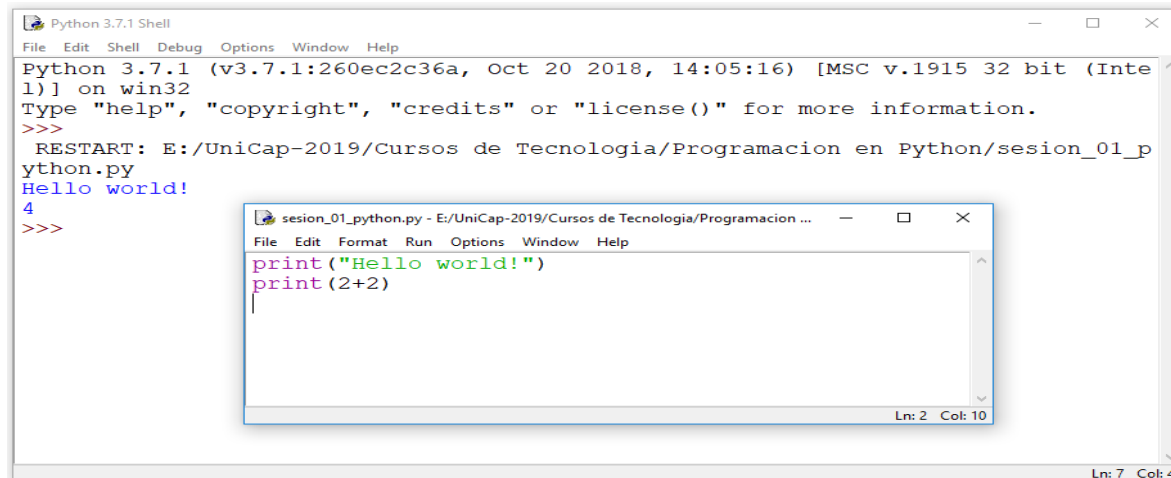
- Algunos influyentes:
  - FORTRAN
    - ciencia / ingenieria
  - COBOL
    - datos de negocios
  - LISP
    - logica e IA
  - BASIC
    - Un lenguaje simple



# Conceptos básicos de programación

- **código** o **código fuente**: La secuencia de instrucciones en un programa.
- **sintaxis**: El conjunto de estructuras y comandos legales que se pueden usar en un lenguaje de programación particular.
- **salida**: Los mensajes impresos al usuario por un programa.
- **consola**: El cuadro de texto en el que se imprime la salida.

Algunos editores de código fuente muestran la consola como una ventana externa, y otros contienen su propia ventana de consola.



The image shows two overlapping windows. The background window is titled 'Python 3.7.1 Shell' and displays the following text:

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: E:/UniCap-2019/Cursos de Tecnologia/Programacion en Python/sesion_01_python.py
Hello world!
4
>>>
```

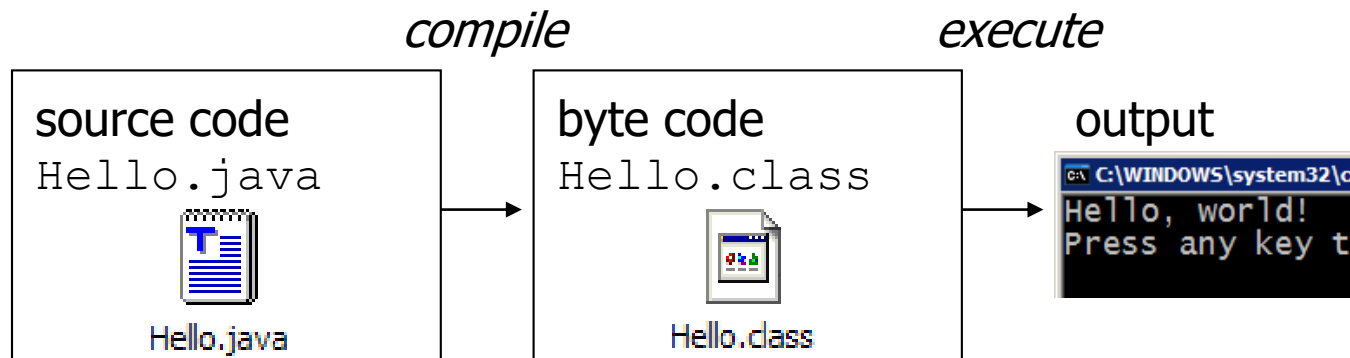
The foreground window is titled 'sesion\_01\_python.py - E:/UniCap-2019/Cursos de Tecnologia/Programacion ...' and displays the following code:

```
print("Hello world!")
print(2+2)
```

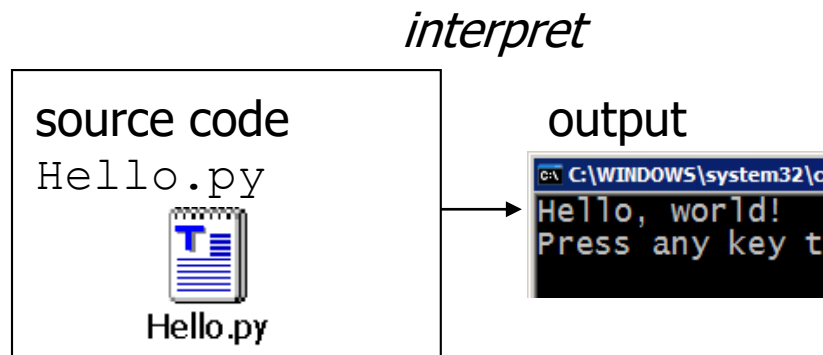
At the bottom right of the foreground window, it says 'Ln: 2 Col: 10'. At the bottom right of the background window, it says 'Ln: 7 Col: 4'.

# Compilación e interpretación

- Muchos lenguajes requieren que compile (traduzca) su programa en una forma que la máquina entienda.



- Python se interpreta directamente en instrucciones de máquina.



# Expresiones

- **expresión:** Un conjunto de operaciones para calcular un valor.

Ejemplos:  $1 + 4 * 3$   
 $42$

- operadores aritméticos que usaremos:

$+$	$-$	$*$	$/$	adicion, sustraccion/negacion, multiplicacion, division
$\%$				modulo, o residuo
$**$				exponenciacion

- **precedencia:** Orden en que se calculan las operaciones.

- $*$   $/$   $\%$   $**$  tienen una precedencia superior que  $+$   $-$

$1 + 3 * 4$  es 13

- Paréntesis: Se puede utilizar para forzar un cierto orden de evaluación.

$(1 + 3) * 4$  is 16

# División entera

- Cuando dividimos enteros con  $/$ , el cociente también es un entero.

$$\begin{array}{r} 3 \\ 4 \overline{) 14} \\ \underline{12} \\ 2 \end{array}$$

$$\begin{array}{r} 52 \\ 27 \overline{) 1425} \\ \underline{135} \\ 75 \\ \underline{54} \\ 21 \end{array}$$

- Mas ejemplos:

- $35 / 5$  es 7
- $84 / 10$  es 8
- $156 / 100$  es 1

- El operador  $\%$  calcula el resto a partir de una división de enteros.

$$\begin{array}{r} 3 \\ 4 \overline{) 14} \\ \underline{12} \\ 2 \end{array}$$

$$\begin{array}{r} 43 \\ 5 \overline{) 218} \\ \underline{20} \\ 18 \\ \underline{15} \\ 3 \end{array}$$

# Números Reales

- Python también puede manipular números reales.
  - Ejemplos: `6.022`      `-15.9997`      `42.0`      `2.143e17`
- Los operadores `+` `-` `*` `/` `%` `**` `()` trabajan para números reales.
- El `/` produce una respuesta exacta: `15.0 / 2.0` es `7.5`
- Las mismas reglas de precedencia también se aplican a los números reales.  
Se evalúa `()` antes que `*` `/` `%` antes que `+` `-`
- Cuando se mezclan enteros y reales, el resultado es un número real. Ejemplo: `1 / 2.0` es `0.5`
  - La conversión se produce por operador.

$$\begin{array}{rcl} 7 / 3 * 1.2 + 3 / 2 & & \\ \underline{2} * 1.2 + 3 / 2 & & \\ 2.4 + 3 / 2 & & \\ 2.4 + \underline{1} & & \\ 3.4 & & \end{array}$$

# Identificadores

- Python tiene algunas reglas sobre cómo se pueden formar los identificadores
- Cada identificador debe comenzar con una letra o un guión bajo, que puede ir seguido de cualquier secuencia de letras, dígitos o guiones bajos.

```
>>> x1 = 10
>>> x2 = 20
>>> y_efecto = 1.5
>>> celsius = 32
>>> 2celsius
File "<stdin>", line 1
    2celsius
      ^
SyntaxError: invalid
syntax
```



# Identificadores

- Python tiene algunas reglas sobre cómo se pueden formar los identificadores
- Identificadores son sensibles a **las mayúsculas**

```
>>> x = 10
>>> X = 5.7
>>> print(x)
10
>>> print(X)
5.7
```

# Identificadores

- Python tiene algunas reglas sobre cómo se pueden formar los identificadores
- Algunos identificadores son parte de Python(a estos se les denominan *palabras reservadas* o *keywords*) y no puede ser utilizado por los programadores como identificadores ordinarios

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	

Palabras clave de Python

# Identificadores

- Python tiene algunas reglas sobre cómo se pueden formar los identificadores.
- Algunos identificadores son parte de Python(a estos se les denominan *palabras reservadas* o *keywords*) y no puede ser utilizado por los programadores como identificadores ordinarios

*Un ejemplo...*

```
>>> for = 4
      File "<stdin>", line 1
        for = 4
          ^
      SyntaxError: invalid
      syntax
```

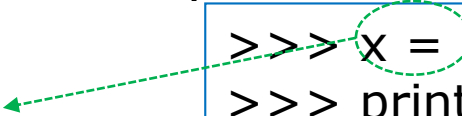
# Comentarios

- Debe iniciar los comentarios con `#`:  
el resto de la línea se ignora.
- Puede incluir una "cadena de documentación" como la primera línea de cualquier nueva función o clase que defina.
- **úsalo:** es un buen estilo incluir uno.
- **def** mi\_funcion (x, y):  
"""Esta es la cadena de documentación. Esta  
La función hace bla, bla, bla"""  
  
# El código iría aquí ...

# Expresiones

- Puede generar nuevos valores de datos (numéricos o de texto) en su programa usando expresiones
- Esta es una expresión que usa el operador de suma.

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```



# Expresiones

- Puede generar nuevos valores de datos (numéricos o de texto) en su programa usando expresiones

- Esta es una expresión que usa el operador de suma.
- Esta es una expresión que usa el operador de *multiplicación*

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

# Expresiones

- Puede generar nuevos valores de datos (numéricos o de texto) en su programa usando expresiones

- Esta es una expresión que usa el operador de suma.

- Esta es una expresión que usa el operador de *multiplicación*

- Esta es otra expresión que usa el operador de suma, pero para concatenar (o pegar) cadenas.

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

# Expresiones

- Puede generar nuevos valores de datos (numéricos o de texto) en su programa usando expresiones

*Otro  
ejemplo...*

```
>>> x = 6
>>> y = 2
>>> print(x - y)
4
>>> print(x / y)
3.0
>>> print(x // y)
3
```

*Otro  
ejemplo...*

```
>>> print(x * y)
12
>>> print(x ** y)
36
>>> print(x % y)
0
>>> print(abs(-x))
6
```



# Expresiones: Resumen de Operadores

Operador	Operación
+	Adición
-	Sustracción
*	Multiplicación
/	Division Exacta
**	Exponenciación
abs()	Valor Absoluto
//	División Entera
%	Residuo

Operaciones numéricas incorporadas de Python

# Literales

- En el siguiente ejemplo, los valores de los parámetros pasados a la función de impresión se denominan técnicamente *literales*
- Más precisamente, "Hola" y "¡Programar es divertido!" se llaman *literales textuales*, mientras 3 y 2.3 se llaman *literales numéricos*

```
>>> print("Hola")
Hola
>>> print("Programar es divertido!")
Programar es divertido!
>>> print(3)
3
>>> print(2.3)
2.3
```

# Variables

- **variable:** Una pieza de memoria con nombre que puede almacenar un valor.

- Uso:
  - Calcular el resultado de una expresión,
  - almacenar ese resultado en una variable,
  - y se usa esa variable mas adelante en el programa.



- **sentencia de asignación :** Almacena un valor en una variable.

- Sintaxis:

***nombre = valor***

- Ejemplos:

$x = 5$

$prom = 3.14$

$x$  5

$prom$  3.14

- Una variable a la que se le ha dado un valor puede usarse en expresiones.

$x + 4$  es 9

- **Ejercicio:** Evaluar la ecuación cuadrática para valores dados  $a$ ,  $b$  y  $c$ .

# Variables

- Una variable es un lugar con nombre en la memoria donde un programador puede almacenar datos y luego recuperarlos utilizando el "nombre" de la variable.
- Los programadores pueden elegir los nombres de las variables.
- Puede cambiar el contenido de una variable en una sentencia posterior.

x = 12.2

y = 14

x = 100

x

~~12.2~~ 100

y

14

# Precedencia de los operadores

La siguiente tabla enumera todos los operadores desde la prioridad más alta a la más baja.

Operador	Descripción
**	Exponenciación (elevar a la potencia)
~ + -	Complemento, suma y resta unaria(nombres de los métodos para los dos últimos son +@ and -@)
* / % //	Multiplicación, división, módulo y división entera
+ -	Adición y sustracción
>> <<	Desplazamiento a la derecha ya la izquierda
&	A nivel de bits 'AND'
^	A nivel de bits exclusivo 'OR' y 'OR' regular
<= < > >=	Operadores de comparación
<> == !=	Operadores de igualdad
= %= /= //= -= += *= **=	Operadores de asignación
is is not	Operadores de Identidad
in not in	Operadores de membresía
not or and	Operadores Lógicos

# Declaraciones de asignación simple

- Se utiliza un literal para indicar un valor específico, que se puede asignar a una *variable*

- x es una variable  
y 2 es su valor

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

# Declaraciones de asignación simple

- Se utiliza un literal para indicar un valor específico, que se puede asignar a una *variable*

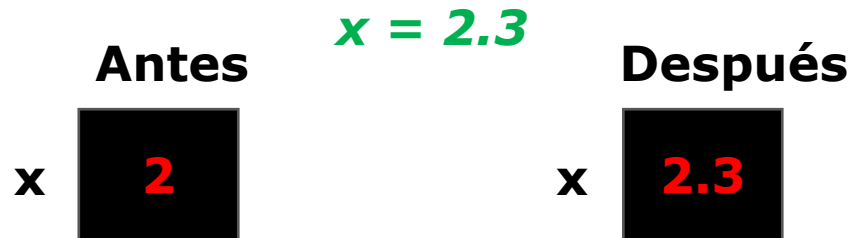
- x es una variable y 2 es su valor
- A x se le pueden asignar diferentes valores; Por lo tanto, se llama una variable.

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

# Sentencias de asignación simple: vista de cuadro

- Una forma sencilla de ver el efecto de una asignación es asumir que cuando una variable cambia, su valor antiguo es reemplazado.

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

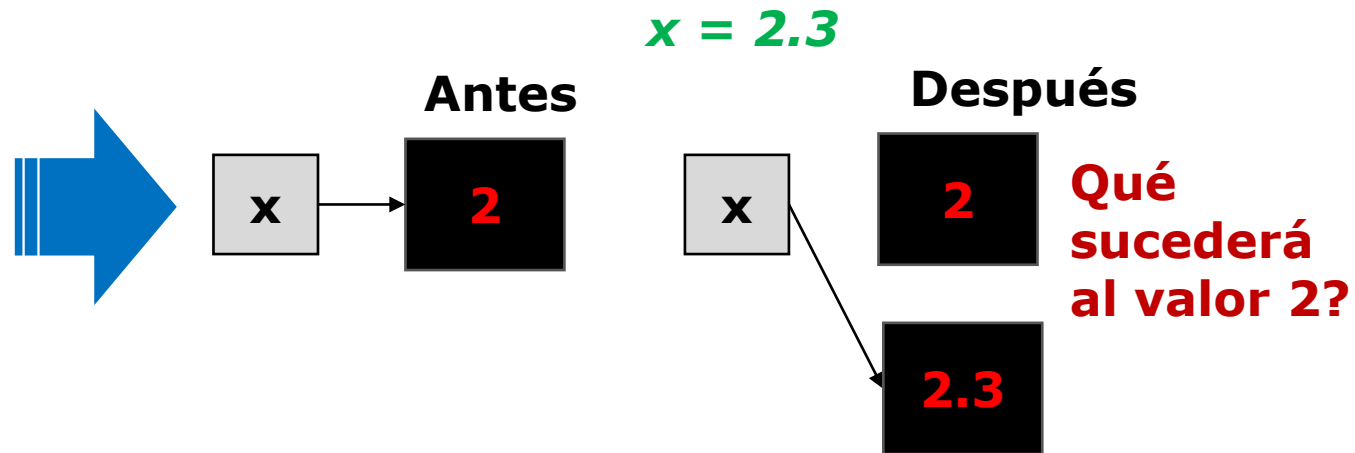




# Declaraciones de asignación simple: Vista real

- Las declaraciones de asignación de Python son en realidad ligeramente diferentes del modelo de "variable como un cuadro"
- En Python, los valores pueden terminar en cualquier parte de la memoria, y se utilizan las variables para referirse a ellos.

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```



# Asignación Simultánea

- Python nos permite también asignar múltiples valores a múltiples variables al mismo tiempo

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
>>>
```

- Esta forma de asignación puede parecer extraña al principio, pero puede resultar muy útil (por ejemplo, para intercambiar valores)

# Asignación Simultánea

- Supongamos que tiene dos variables **x** e **y**, y desea intercambiar sus valores (es decir, desea que el valor almacenado en **x** esté en **y** y viceversa)

```
>>> x = 2
>>> y = 3
>>> x = y
>>> y = x
>>> x
3
>>> y
3
```

**X No se puede hacer con dos tareas simples**

# Asignación Simultánea

- Supongamos que tiene dos variables **x** e **y**, y desea intercambiar sus valores (es decir, desea que el valor almacenado en **x** esté en **y** y viceversa)

Hasta ahora, hemos estado usando diferentes nombres para las variables. Estos nombres son técnicamente llamados **identificadores**

```
>>> x = 2
>>> y = 3
>>> temp = x
>>> x = y
>>> y = temp
>>> x
3
>>> y
2
>>>
```

**Se puede hacer con tres tareas simples, pero mas eficientemente con asignación simultánea**

# Tipos de Datos

- En Python, todos los datos tienen un **"Tipo"** de dato asociado.
- Puede encontrar el "Tipo" de cualquier dato utilizando la función `type()`:

`type("Hola!")` produce `<type 'str'>`

`type(True)` produce `<type 'bool'>`

`type(5)` produce `<type 'int'>`

`type(5.0)` produce `<type 'float'>`

- Tenga en cuenta que Python admite dos tipos diferentes de números, **enteros** (`int`) y números de **punto flotante** (`float`). Los números de punto flotante tienen una parte fraccionaria (dígitos después del lugar decimal), mientras que los enteros no.

# print

- `print` : Produce salida de texto en la consola.

- Sintaxis:

```
print("Mensaje")
```

```
print(Expresion)
```

- Imprime el mensaje de texto o el valor de expresión en la consola y mueve el cursor hacia abajo a la siguiente línea.

```
print(Item1, Item2, ... , ItemN)
```

- Imprime varios mensajes y / o expresiones en la misma línea.

- Ejemplos:

```
print("Hello, world!")
```

```
edad = 25
```

```
print("Tu tienes ", 65 - edad, " años hasta la jubilación")
```

## Salida:

```
Hello, world!
```

```
Tu tienes 40 años hasta la jubilación
```

# Asignando entrada

- Hasta ahora, hemos estado usando valores especificados por los programadores e impresos o asignados a variables. ¿Cómo podemos permitir que los usuarios (no los programadores) ingresen valores?
- En Python, la entrada se realiza a través de una instrucción de asignación combinada con una función integrada llamada **input**

■ **<variable> = input(<texto>)**

Cuando Python encuentra una llamada a **input**, imprime **<texto>** (que es un literal de cadena), luego se detiene y espera a que el usuario escriba un texto y presione la tecla <Enter>

# input

- **input** : Lee un número de la entrada del usuario.
  - Puede asignar (almacenar) el resultado de la entrada en una variable.

- **Ejemplo:**

```
edad = input("Ingresa tu edad: ")
print("Tu edad es: ", edad)
print("Tienes ", 65 - edad, " años antes de jubilarte")
```

Salida:

```
Ingresa tu edad: 53
Tu edad es: 53
Tienes 12 años antes de jubilarte
```

- **Ejercicio:** Escriba un programa de Python que le pida al usuario su cantidad de dinero, luego informa cuántos Nintendo Wiis puede pagar la persona y cuánto más dinero necesitará para pagar un Wii adicional.



# Asignando entrada

- Aquí hay una muestra de interacción con el intérprete de Python:  
Observe que todo lo que escriba el usuario se almacena como una cadena
- ¿Qué pasa si el usuario ingresa un número?

```
>>> name = input("Ingrese su nombre: ")
Ingrese su nombre : Victor
>>> name
'Victor'
>>>
```

# Asignando entrada

- Aquí hay una muestra de interacción con el intérprete de Python:
- ¿Cómo podemos obligar a que un número de entrada se almacene como un número y no como una cadena?
- Podemos usar la función **eval** incorporada, que puede "envolver" a la función input

```
>>> numero = input("Ingrese un numero: ")
Ingrese un numero : 3
>>> numero
'3'
>>>
```

**Aun un string!** ←



# Asignando entrada

- Aquí hay una muestra de interacción con el intérprete de Python:

```
>>> number = eval(input(" Dame un numero : "))
Dame un numero : 3
>>> number
3
>>>
```

**Ahora un int  
(sin comillas simples)!**

# Asignando entrada

- Aquí hay una muestra de interacción con el intérprete de Python:

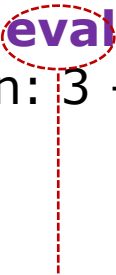
```
>>> number = eval(input("Dame un numero: "))
Dame un numero: 3.7
>>> number
3.7
>>>
```

**Y ahora un float  
(sin comillas simples)!**

# Asignando entrada

- Aquí hay otro ejemplo de interacción con el intérprete de Python:

```
>>> number = eval(input("Ingrese ecuacion: "))  
Ingrese ecuacion: 3 + 2  
>>> number  
5  
>>>
```



A red dashed oval highlights the word 'eval' in the first line of code. A red dashed arrow points from the 'eval' function to the input string '3 + 2' in the second line. Another red dashed arrow points from the input string '3 + 2' to the variable 'number' in the third line.

**La función eval evaluará esta fórmula y devolverá un valor, que luego se asigna a la variable "número"**

# Conversión de tipos de Dato

- Además, podemos convertir la cadena de salida de la función `input` en un entero o un flotante utilizando las funciones incorporadas **`int`** y **`float`**

**Un entero  
(sin comillas simples)!**

```
>>> number = int(input("Ingrese un numero: "))
Ingrese un numero: 3
>>> number
3
>>>
```

# Conversión de tipos de Dato

- Además, podemos convertir la cadena de salida de la función `input` en un entero o un flotante utilizando las funciones incorporadas **`int`** y **`float`**

**Un decimal**

**(sin comillas simples)!**

```
>>> number = float(input("Ingrese un numero: "))
Ingrese un numero: 3.7
>>> number
3.7
>>>
```

# Conversión de tipos de Dato

- De hecho, podemos hacer varios tipos de conversiones entre cadenas, enteros y flotantes usando las funciones incorporadas `int`, `float` y `str`.

```
>>> x = 10
>>> float(x)
10.0
>>> str(x)
'10'
>>>
```

```
>>> y = "20"
>>> float(y)
20.0
>>> int(y)
20
>>>
```

```
>>> z = 30.0
>>> int(z)
30
>>> str(z)
'30.0'
>>>
```

integer → float  
integer → string

string → float  
string → integer

float → integer  
float → string



# Conversión explícita e implícita de tipos de datos

- La conversión de datos puede ocurrir de dos maneras en Python:
- **Conversión explícita de datos**(esto lo vimos anteriormente con las funciones int, float y str incorporadas)
- **Conversión explícita de datos**(Se lleva a cabo automáticamente durante el tiempo de ejecución entre SOLO valores numéricos.
- **Ejm.**, Agregar un valor flotante y un entero dará como resultado automáticamente un valor flotante.
- Por ejemplo, agregar una cadena y un entero (o un flotante) generará un error ya que la cadena no es numérica.
- Aplica la promoción de tipo para evitar la pérdida de información
  - La conversión va de entero a flotante (por ejemplo, al agregar un flotante y un entero) y no al revés, ya que la parte fraccionaria del flotante no se pierde

# Conversión Tipos de Datos Implícitos: Ejemplos

El resultado de una expresión que involucra un número flotante junto con (un) número entero es un número flotante

```
>>> print(2 + 3.4)
5.4
>>> print( 2 + 3)
5
>>> print(9/5 * 27 +
32)
80.6
>>> print(9//5 * 27
+ 32)
59
>>> print(5.9 + 4.2)
10.1000000000000001
>>>
```

# Conversión Tipos de Datos Implícitos: Ejemplos

- El resultado de una expresión que involucra un número flotante junto con (un) número entero es un número flotante
- El resultado de una expresión que involucra valores del mismo tipo de datos no resultará en ninguna conversión

```
>>> print(2 + 3.4)
5.4
>>> print( 2 + 3)
5
>>> print(9/5 * 27 + 32)
80.6
>>> print(9//5 * 27 + 32)
59
>>> print(5.9 + 4.2)
10.100000000000001
>>>
```

# Comandos Math

- Python tiene comandos útiles para realizar cálculos.

Nombre del comando	Descripción
<code>abs(<b>valor</b>)</code>	valor absoluto
<code>ceil(<b>valor</b>)</code>	redondea
<code>cos(<b>valor</b>)</code>	coseno, en radianes
<code>floor(<b>valor</b>)</code>	redondea hacia abajo
<code>log(<b>valor</b>)</code>	logaritmo, base e
<code>log10(<b>valor</b>)</code>	logaritmo, base 10
<code>max(<b>valor1</b>, <b>valor2</b>)</code>	mayor de dos valores
<code>min(<b>valor1</b>, <b>valor2</b>)</code>	menor de dos valores
<code>round(<b>valor1</b>)</code>	número entero más cercano
<code>sin(<b>valor1</b>)</code>	seno, en radianes
<code>sqrt(<b>valor1</b>)</code>	raíz cuadrada

Constante	Descripción
e	2.7182818...
pi	3.1415926...

- Para usar muchos de estos comandos, debe escribir lo siguiente en la parte superior de su programa Python:

```
from math import *
```