UNIVERSIDAD DE MÁLAGA

FGUMA FUNDACIÓN GENERAL UNIVERSIDAD DE MÁLAGA

CURSOS ONLINE

FUNDACIÓN GENERAL UNIVERSIDAD DE MÁLAGA

Variables, Expresiones y Sentencias

Unidad 2

Python para principiantes

dgm@uma.es

Constantes

- Los **valores fijos** tales como **números**, **letras** y **cadenas de caracteres (strings)** son conocidos como **“constantes”** porque sus valores no cambian
- Las constantes numéricas se representan tal cual (como **números**)
- Las constantes de cadenas caracteres utilizan **comillas simples (‘) o dobles (")**

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Palabras Reservadas

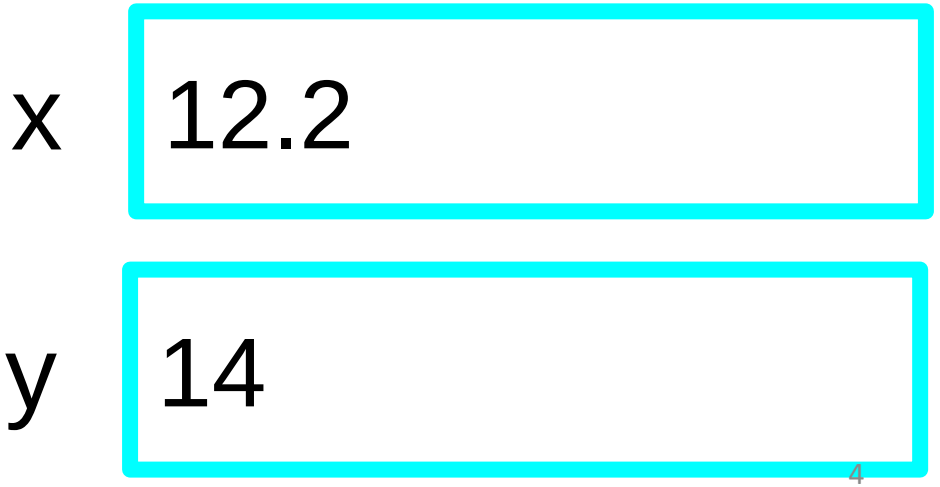
No se pueden utilizar las palabras reservadas como nombres de variables / identificadores

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

Variables

- Una variable es un lugar con "**nombre**" en la memoria donde un programador puede **almacenar datos** y **recuperarlos más tarde** usando el nombre de la variable
- Los programadores eligen los nombres de las variables
- El contenido de una variable se puede cambiar con la **sentencia de asignación**

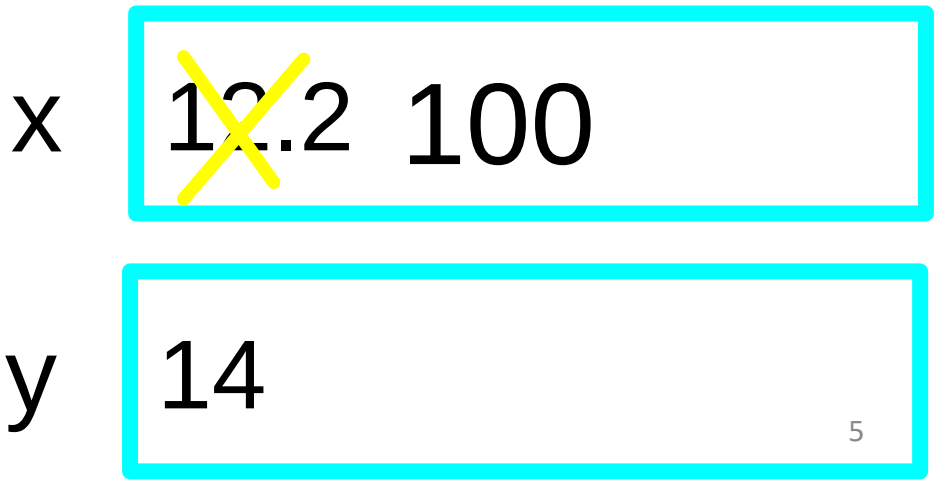
```
x = 12.2
y = 14
```



Variables

- Una variable es un lugar con "**nombre**" en la memoria donde un programador puede **almacenar datos** y **recuperarlos más tarde** usando el nombre de la variable
- Los programadores eligen los nombres de las variables
- El contenido de una variable se puede cambiar con la **sentencia de asignación**

```
x = 12.2
y = 14
x = 100
```



Reglas para los Nombres de Variables en Python

- Deben comenzar con una letra o subrayado _
- Solo pueden contener letras, números y subrayados
- Se **distinguen** mayúsculas de minúsculas

```
Bien:      spam    eggs    spam23      _speed
Mal:       23spam      #sign   var.12
Diferentes:     spam    Spam    SPAM
```

Mnemónicos para Nombres de Variables

- Dado que son los programadores quienes eligen los nombres, hay unas “**mejores prácticas**”

- Elegir un buen nombre es fundamental para que se entienda tu código
- Los nombres de las variables tienen que ayudar a recordar lo que queremos que almacenen (“mnemónico” = “ayuda a recordar”)

http://en.wikipedia.org/wiki/Mnemonic

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

¿Qué hace este fragmento de código?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

¿Qué hacen estos fragmentos de código?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

¿Qué hacen estos fragmentos de código?

```
horas = 35.0
precio = 12.50
total = horas * precio
print(total)
```

Sentencias o Líneas

```
x = 2
x = x + 2
print(x)
```

Sentencia de Asignación
Asignación con expresión
Sentencia print

Variable Operador Constante Función

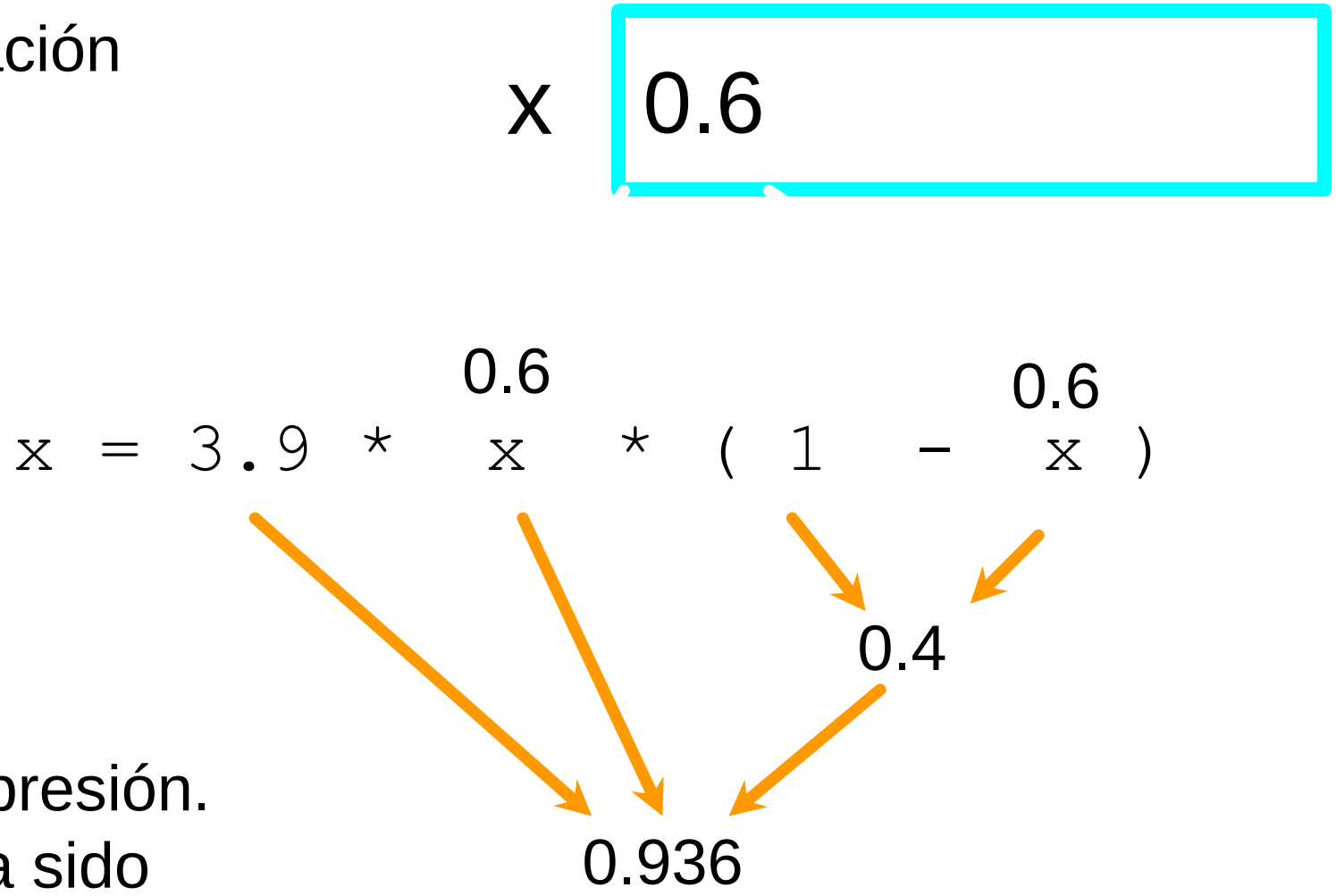
Sentencias de Asignación

- Asignamos un valor a una variable usando la sentencia de asignación (=)

- Una sentencia de asignación consiste en una **expresión** en la parte derecha y una **variable** en la parte izquierda para almacenar el resultado

```
x = 3.9 * x * ( 1 - x )
```

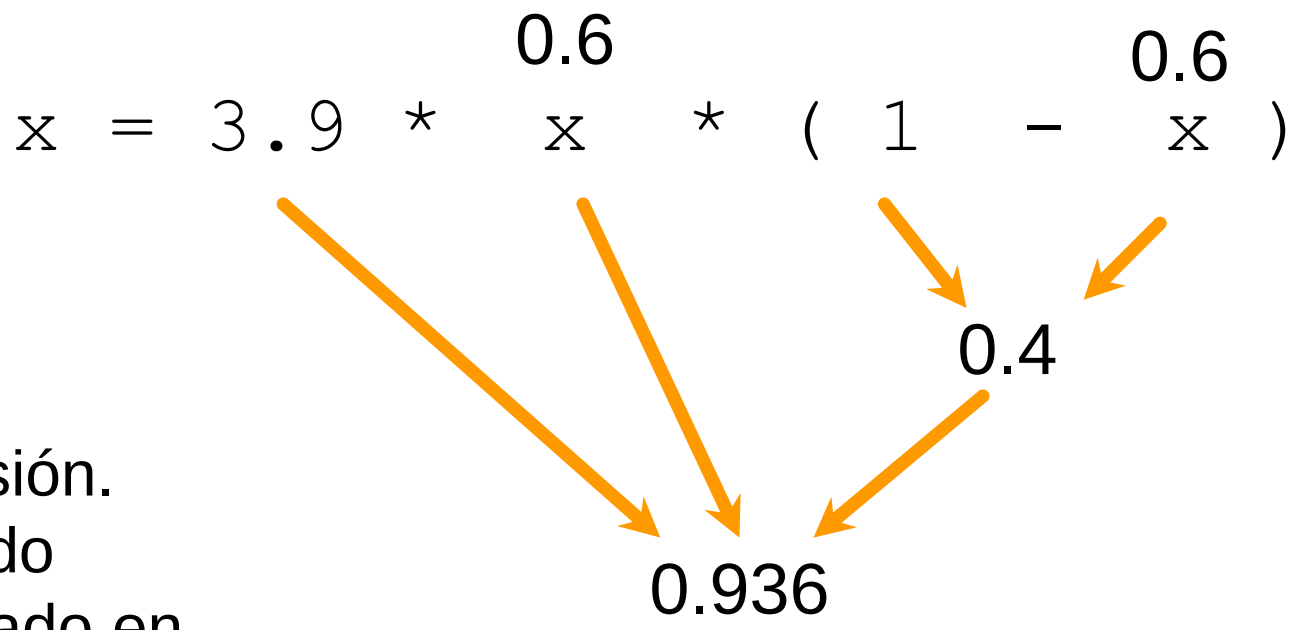
Una variable es una localización en memoria utilizada para almacenar un valor (0.6)



La parte derecha es una expresión. Una vez que la expresión ha sido evaluada, el resultado es colocado en (asignado a) x.

El valor almacenado en una variable puede ser **actualizado** reemplazando el antiguo valor (0.6) con uno nuevo (0.936).

x 0.6 0.936



La parte derecha es una expresión. Una vez que la expresión ha sido evaluada, el resultado es colocado en (asignado a) x.

Expresiones…

Expresiones Numéricas

- Debido a la falta de símbolos matemáticos en los teclados, se usan algunos símbolos específicos para expresar las operaciones matemáticas clásicas
- Por ejemplo:
  - El asterisco es la multiplicación
  - Las potencias (elevar un número a una potencia) utilizan un doble **

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia
%	Resto

Expresiones Numéricas

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia
%	Resto

Orden de Evaluación

- Cuando usamos varios operadores juntos Python debe saber **qué hacer primero**
- Esto es lo que se llama "**precedencia** de operadores"
- ¿Qué operador "toma precedencia" sobre los otros?

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

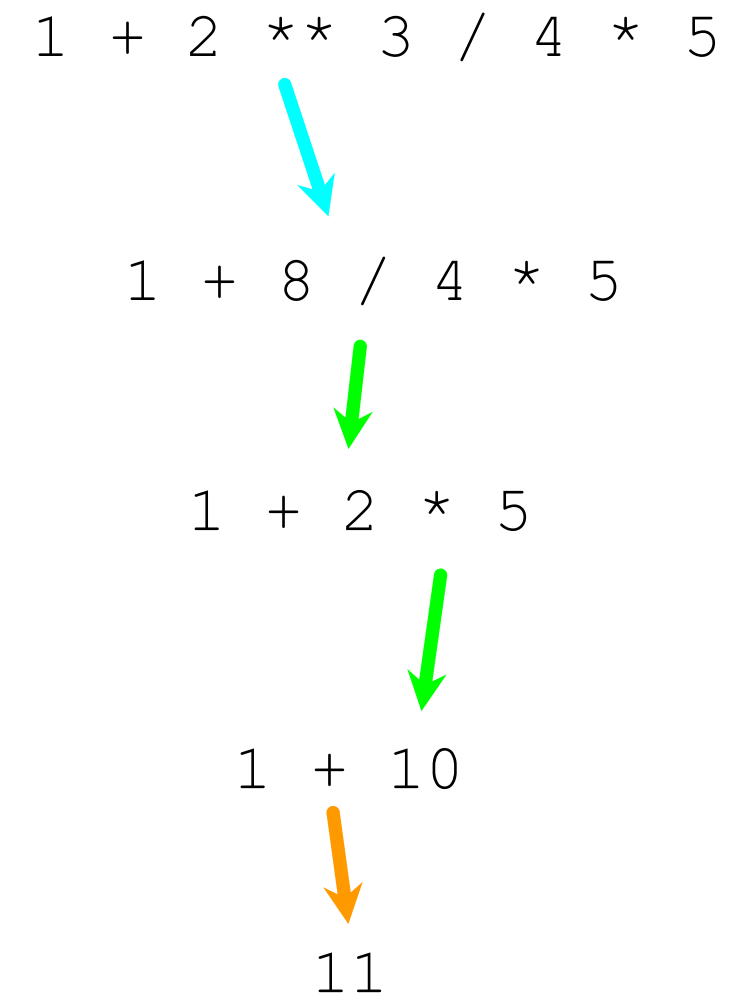
Reglas de Precedencia de Operadores

- De la precedencia más alta a la más baja:
  - Los paréntesis son siempre respetados
  - Exponenciación (elevar a una potencia)
  - Multiplicación, División y Resto
  - Suma y Resta
  - De izquierda a derecha

Paréntesis
Potencia
Multiplicación
Suma
Izquierda a Derecha

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Paréntesis
Potencia
Multiplicación
Suma
Izquierda a
Derecha



Precedencia de Operadores

- Recordar las reglas anteriores
- Utilizar paréntesis **en caso de duda**
- Mantener las expresiones matemáticas lo suficientemente **simples** para que sean fáciles de entender
- “Romper” series largas de operaciones matemáticas para hacerlas más claras

Paréntesis
Potencia
Multiplicación
Suma
Izquierda a Derecha

Tipos…

¿Qué significa el “Tipo”?

- En Python las variables y constantes tienen un **“tipo”**
- Python conoce la diferencia entre un número entero y una cadena de caracteres (string)
- Por ejemplo “+” significa “sumar” si algo es un número y “concatenar” si algo es una cadena de caracteres

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

concatenar = poner junto

Cuestiones con Tipos

- Python conoce el “tipo” de todo
- Algunas operaciones están prohibidas
- No se puede “sumar 1” a un string
- Podemos preguntar a Python el tipo de algo usando la función **type()**

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```

Varios Tipos de Números

- Los números tienen dos tipos principales
  - Los número **enteros**: -14, -2, 0, 1, 100, 401233
  - Los números **flotantes** (con decimales): -2.5 , 0.0, 98.6, 14.0
- Hay otros tipos de números – variaciones de flotantes y enteros

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```

Conversiones de Tipo

- Cuando se ponen enteros y flotantes en una expresión, el entero es **convertido** implícitamente a flotante
- Este comportamiento se puede controlar con las funciones **int()** y **float()**

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

División Entera

- La división de enteros produce como resultado un número flotante

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

Esto era diferente en Python 2.x

Conversiones de Strings

- Se puede utilizar int() y float() para convertir entre cadenas de caracteres y enteros
- Se obtendrá un error si el string no tiene un número “bien escrito”

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object
to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'x'
```

Entrada de Datos

- Podemos hacer que Python lea datos del usuario utilizando la función **input()**
- La función input() retorna una cadena de caracteres

```
nam = input('Who are you? ')
print('Welcome', nam)
```

Who are you? Chuck
Welcome Chuck

Convirtiendo la Entrada del Usuario

- Si queremos leer un número del usuario, debemos **convertir de string a número** utilizando la función correspondiente
- Después veremos que hacer con datos de entrada incorrectos

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

Europe floor? 0
US floor 1

Comentarios en Python

- Todo lo que se escriba después de un # es **ignorado** por Python
  - Ojo, excepto si estamos dentro de una cadena de caracteres.
    - P.ej. “hola # aquí #“
- ¿Por qué comentar?
  - - Para **describir** lo que va a ocurrir en una secuencia de código
  - **Documentar** quién escribió el código u otra información auxiliar
  - **Desactivar** una línea de código – quizás temporalmente

```
# Obtener el nombre del archivo y abrirlo
name = input('Nombre fichero:')
handle = open(name, 'r')

# Contar las apariciones de palabras
counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

# Encontrar la palabra más común
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

# Todo hecho
print(bigword, bigcount)
```

Resumen

- Tipos
- Palabras reservadas
- Variables (mnemónicos)
- Operadores
- Precedencia de operadores
- División entera
- Conversión entre tipos
- Entrada de usuario
- Comentarios (#)

Ejercicio

Escribir un programa que pregunte al usuario un número de horas y el precio por hora para calcular el total a pagar.

```
Introduzca las horas: 35
Introduzca el precio/hora: 2.75

Total: 96.25
```

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here
Spanish Version: Daniel Garrido (dgm@uma.es)

Continue…