



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

2021/2022

Programación

TEMA 2

Tema 2.
Valores,
variables, tipos,
operadores y
expresiones

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
3. Variables
4. Operador de asignación
5. Conversión de tipo
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres
7. Formato de salida
8. Documentación de código
9. Indexación y funciones de cadenas o strings



Lectura y ejercicios (en PoliformaT)

Tema2-teoria-Python-reader.pdf

Programación en Python TEMA 2

Valores, variables, tipos, operadores y expresiones.

Universidad Politécnica de Valencia

Índice

2021-2022

1	Valores y tipos	2
2	Variables	3
3	Nombres de variables y palabras claves	3
4	Instrucciones	4
5	Operadores y operandos	4
6	Expresiones	5
7	Orden de las operaciones	6
8	Operador módulo	6
9	Operaciones con cadenas	7
10	Comentarios y documentación	7
11	Elección de nombres de variables mnemónicos	8
12	Errores de sintaxis común a principiantes	9
13	Petición de información al usuario (la instrucción <code>input</code>)	10
14	Más sobre la función <code>print</code> y las cadenas o <code>strings</code> .	11
15	Formateando la salida: la función <code>format</code>	12
16	Más sobre <code>strings</code>	14
	Ejercicios de respuesta abierta	19
	Ejercicios de tipo test	21

10.1007_s12293-018-0263-8-citation.ris

El contenido de este boletín está basada en material de diferentes libros open source:

- *Python for everybody*, Copyright 2009 - Charles Severance.
- *Think Python: How to Think Like a Computer Scientist*, Copyright 2015 - Allen Downey (Traducción de Jorge Espinoza).

Ambos trabajos están registrados bajo una Licencia Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. Este licencia está disponible en <http://creativecommons.org/licenses/by-nc-sa/3.0/>

1. Tipos de datos

- Cuando se programa en Python, se ha de tener siempre presente qué tipo de datos tu programa está procesando
- Un **tipo de dato** es un conjunto de valores y de operaciones definidas sobre esos valores
- El lenguaje Python permite trabajar con varios tipos de datos. En este tema presentaremos únicamente los más básicos:

Tipo de dato	Conjunto de valores	Operadores básicos	Ejemplos
int	Números enteros	+ - * // % **	99 12 2147483647
float	Números reales	+ - * / **	3.14 2.5 6.022e23
bool	Verdadero/falso	and or not == != > < >= <=	True False
str	Cadenas de caracteres o strings	+	'AB' 'Hola' '2.5' "Hello" "where?"

1. El tipo importa

- Python conoce el «tipo» de todos los datos
- Algunas operaciones no están permitidas
- No puedes «añadir 1» a una cadena (string)
- Podemos consultarle a Python el tipo de un dato utilizando la función **type()**

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
TypeError: cannot concatenate 'str'
and 'int' objects
>>> type(eee)
<type 'str'>
>>> type('hello')
<type 'str'>
>>> type(1)
<type 'int'>
>>>
```

Índice

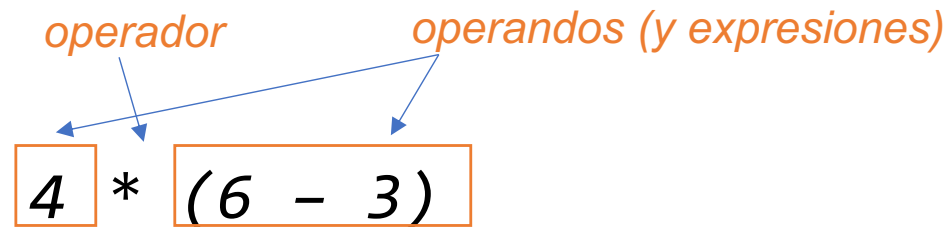
TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
- 2. Expresiones y operadores (aritméticos, relacionales, lógicos)**
3. Variables
4. Operador de asignación
5. Conversión de tipo
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres
7. Formato de salida
8. Documentación de código
9. Indexación y funciones de cadenas o strings



2. Expresiones

- Una **expresión** es una combinación de valores, variables y operadores que Python evalúa para producir un valor resultado
- Muchas expresiones tienen el aspecto de fórmulas matemáticas:



- Python usa unas reglas de precedencia bien definidas para aplicar los operadores
 - Para operaciones aritméticas, la multiplicación y la división se calculan antes que las sumas y restas
 - Pueden usarse paréntesis para modificar el orden de las operaciones
 - **papomudisure** (**p**aréntesis, **p**otencias, **m**ultiplicación, **d**ivisión, **s**uma, **r**esta)

2. Operadores aritméticos

Operador	Operación	Comentario
+ , -	Signo	Cambio de signo
+	Suma	Suma de números reales o enteros
-	Resta	Resta de números reales o enteros
*	Multiplicación	Multiplicación de números reales o enteros
//	División entera	División de números enteros, descartando la parte fraccional
%	Resto	Resto de la división entera
/	División	División de números reales
**	Potencia	Elevar la base al exponente

2. Ejemplos de expresiones aritméticas enteras (int)

Expresión entera	Valor	Comentario
+99	99	Signo positivo
-99	-99	Signo negativo
--99	99	Signo negativo
5 + 3	8	Suma
5 - 3	2	Resta
5 * 3	15	Multiplicación
5 // 3	1	Sin parte fraccional
5 % 3	2	Resto
5 ** 3	125	Potencia
5 // 0	ZeroDivisionError	Error: división por cero
3 * 5 - 2	13	* tiene precedencia
3 + 5 // 2	5	// tiene precedencia
3 - 5 - 2	-4	Asociativo por la izquierda
(3 - 5) - 2	-4	Mejor estilo
3 - (5 - 2)	0	No ambiguo
2 ** 2 ** 3	256	Asociativo por la derecha
(2 ** 2) ** 3	64	Cambia la expresión
2 ** (2 ** 3)	256	Mejor estilo, no ambiguo

2. Ejemplos de expresiones aritméticas reales (float)

Expresión real	Valor	Comentario
3.141 + 2.0	5.141	Suma
3.141 - 2.0	1.141	Resta
3.141 * 2.0	6.282	Multiplicación
3.141 / 2.0	1.5705	División
5.0 / 3.0	1.6666666666666667	Los números reales tienen 17 dígitos de precisión como máximo
3.141 ** 2.0	9.865881	Potencia
1.0 / 0.0	ZeroDivisionError	Error: división por cero
2.0 ** 10000.0	OverflowError	Error: no se puede representar un número real tan grande

2. Ejemplos de expresiones aritméticas reales

- ❑ Al contrario que los números enteros, los números float tienen un tamaño y una precisión limitados y entonces son aproximaciones.
- ❑ Los números float se representan en el hardware de la computadora como fracciones de base 2 (binarias).
- ❑ Eso hace que no puede representar de forma precisa números como 0.1, 0.2 o 0.3 de ninguna manera.
- ❑ Es necesario tenerlo en cuenta, ya que una expresión como $0.1 + 0.1 + 0.1$ devuelve el inesperado resultado de 0.30000000000000004
- ❑ Es porque 0.1 no realmente es exactamente 0.1..... es una aproximación
- ❑ Si Python imprimiera el verdadero valor decimal de la aproximación binaria almacenada para 0.1, tendría que mostrar

```
>>>>> 0.1
```

```
0.1000000000000000055511151231257827021181583404541015625
```

- ❑ Mirar por ejemplo:
 - <https://docs.python.org/3/tutorial/floatingpoint.html>
 - <http://puntoflotante.org/>

2. Cadenas de caracteres o strings

- Las cadenas de texto son secuencias de caracteres (letras, dígitos, símbolos, espacios y caracteres de control)
- Se llaman **strings** en ingles
 - Se representan entre comillas simples (') o doubles (")
 - Usar la contrabarra para representar caracteres especiales:

Secuencia	Significado
'\t'	Tabulador
'\n'	Salto de línea
'\''	Comilla simple
'\\'	Backslash

- Ejemplos

Expresión	Valor
"Tanja's computer"	"Tanja's computer"
'Tanja\'s computer'	"Tanja's computer"
"99'	Syntax error
"appel"	'appel'

2. Operador (+) de cadenas de caracteres o strings

- Es posible **concatenar** (o **unir**) dos cadenas de texto mediante el operador +

Expresión	Valor
'Hola, ' + 'Mundo'	'Hola, Mundo'
'123' + '456'	'123456'
'1234' + ' ' + ' ' + '99'	'1234 + 99'
'123' + 456	TypeError: can only concatenate str (not "int") to str
"appel"	'appel'

2. Operador (*) de cadenas de caracteres o strings

- Hay un operador de **repetición** de cadenas
 - El símbolo que lo denota es *, el mismo que para la multiplicación de enteros y/o reales
 - El resultado es la concatenación de la cadena consigo misma, tantas veces como indique el número entero:

Expresión	Valor
'Hola' * 5	'HolaHolaHolaHolaHola'
20 * '-'	'-----'
6 * '=' + 3 * '(0)' + 6 * '='	'=====(0)(0)(0)====='

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
- 3. Variables**
4. Operador de asignación
5. Conversión de tipo
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres
7. Formato de salida
8. Documentación de código
9. Indexación y funciones de cadenas o strings



3. Variables

- Una **variable** es un nombre asociado a un tipo de dato
- Los programas utilizan variables para almacenar valores
- El **nombre de una variable** es una secuencia de letras, dígitos y subrayados (_)
- El nombre lo elige el programador
 - El primer carácter del nombre no puede ser un dígito
 - Python distingue entre letras minúsculas y mayúsculas
 - Las palabras reservadas por el lenguaje y no pueden usarse como nombres de variables
 - Ejemplos:
 - abc
 - AB_
 - abc123
 - a_b

3. Palabras reservadas para Python

- No se pueden usar como nombre de variables

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	

3. Reglas de estilo para nombres de variables

- Los programadores siguen normalmente unas reglas de estilo a la hora de dar nombre a sus variables:
 - Deben comenzar por una letra minúscula, seguida de otras letras (mayúsculas o minúsculas) y dígitos
 - Se usan las mayúsculas para delimitar las palabras en variables compuestas por varias de ellas
 - Ejemplos: *i, x, y, total, esBisiesto, fechaDeNacimiento*

3. Reglas mnemotécnicas para nombres de variables

- Puesto que los programadores elegimos el nombre de las variables, es mejor «practicar un poco»
- Nombramos las variables para recordar qué vamos a almacenar en ellas («mnemotecnia» = «ayuda para memorizar»)

<http://es.wikipedia.org/wiki/Mnemotecnia>

3. Reglas mnemotécnicas para nombres de variables

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

¿Qué hace este código?

3. Reglas mnemotécnicas para nombres de variables

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

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

¿Qué hace este
código?

3. Reglas mnemotécnicas para nombres de variables

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

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

¿Qué hace este
código?

```
hours = 35.0  
rate = 12.50  
pay = hours * rate  
print(pay)
```

3. Constantes

- Una variable se considera **constante** cuando su valor es siempre el mismo
- La convención, en este caso, es escribir su nombre mediante mayúsculas y dígitos, y usar el subrayado como separador de palabras
 - Ejemplos:
 - *VELOCIDAD_DE_LA_LUZ*
 - *AZUL_MARINO*

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
3. Variables
- 4. Operador de asignación**
5. Conversión de tipo
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres
7. Formato de salida
8. Documentación de código
9. Indexación y funciones de cadenas o strings



4. El operador de asignación

- Para definir una variable en Python usamos el operador de asignación (=)

- Ejemplo: al escribir

```
algun_variable = 1234
```

estamos:

- Definiendo `algun_variable` como una nueva variable
- Asignándole a la variable a el valor entero 1234
- El lado derecho de la asignación puede ser cualquier tipo de expresión
 - Python evalúa la expresión y le asigna el resultado a la variable
 - Ejemplo:

	a	b	c
a = 1234	1234		
b = 99	1234	99	
c = a + b	1234	99	1333

4. Ejemplo: incremento de una variable

- El siguiente código asigna el valor 17 a la variable *i*, y luego incrementa en uno su valor:

```
i = 17
```

```
i = i + 1
```

- Esta es una operación muy común en programación. Los pasos son los siguientes:
 - Crea la variable *i* y le asigna el valor entero 17
 - Python evalúa la expresión *i* + 1 y obtiene el valor 18
 - Asigna este nuevo valor, 18, a la variable *i*

	<u><i>i</i></u>
<i>i</i> = 17	17
<i>i</i> = <i>i</i> + 1	18

4. Asignaciones con operador

- Incrementar el valor de una variable es tan frecuente que existe una forma compacta de hacerlo en Python
- La instrucción `i = i + 1` puede escribirse como: `i += 1`
- Ejemplo:

```
a = 3  
b = 2  
a += 4 * b
```
- Significa: `a = a + 4 * b`
- El valor final de `a` será 11

4. Asignaciones con operador

- Todos los operadores aritméticos tienen su asignación con operador asociada:

Asignación	Valor de a
a = 1	1
a += 10	11
a -= 7	4
a *= 4	16

Asignación	Valor de a
a //= 3	5
a %= 2	1
a /= 2	0.5
a **= 2	0.25

4. Ejemplo: intercambio del valor de dos variables

- El siguiente código intercambia los valores de las variables a y b:

a = 1234

b = 99

t = a

a = b

b = t

t = a

a = b

b = t

a	b	t
1234	99	
1234	99	1234
99	99	1234
99	1234	1234

- Los pasos son los siguientes:
 - a y b tienen los valores 1234 y 99 respectivamente
 - t = a
 - asigna a t el valor de a (ambos comparten ahora el mismo valor: 1234)
 - a = b
 - asigna el valor de b a a (99). El valor que almacena una variable se pierde cuando se le asigna uno nuevo
 - b = t
 - asigna el valor de t a b (1234)

ejercicios 1 hasta 4

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
3. Variables
4. Operador de asignación
- 5. Conversión de tipo**
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres
7. Formato de salida
8. Documentación de código
9. Indexación y funciones de cadenas o strings



5. Conversión de tipo

- Los programas manejan normalmente distintos tipos de datos
- Es necesario prestar atención al tipo de cada dato para escoger los valores y las operaciones a realizar con él
- Python convierte automáticamente los enteros a reales cuando se requiere (**conversión implícita**)
 - Ejemplo: $10/4$ se evalúa como 2.5 Python convierte el resultado a un float porque el resultado de dividir dos números es un número float
 - Ejemplo: $3 + 4.0$ se evalúa a 7.0. Python ha convertido implícitamente el integer 3 a un float 3.0
- Otra posibilidad es hacer una **conversión explícita** usando una función de conversión (que toma un argumento de un tipo y devuelve uno de otro tipo)
 - Estas funciones tienen el mismo nombre que los tipos de datos que hemos visto: **int()**, **float()**, **bool()** y **str()**

5. Conversión de tipo explícita

Función	Uso	Ejemplo	Res.	Explicación
int()	Convierte el argumento a un número entero	int(2.7) int(True) int('10')	2 1 10	Descarta los decimales <i>True</i> = 1 y <i>False</i> = 0 Convierte la cadena en entero
round()	Redondea un número real a entero	round(2.4) round(2.6)	2 3	Redondea el número real al entero más cercano
float()	Convierte el argumento a un número real	float(2) float(True) float('2.5')	2.0 1.0 2.5	Entero a número real <i>True</i> = 1.0 y <i>False</i> = 0.0 Convierte la cadena en real
bool()	Convierte el argumento a un booleano	bool(0) bool(1.0) bool('')	False True False	0=False, distinto de 0=True 0.0=False, distinto de 0.0=True String vacío=False, no vacío=True
str()	Convierte el argumento a una cadena de texto	str(10) str(2.5) str(True)	'10' '2.5' 'True'	Convierte el entero en cadena Convierte el real en cadena Convierte el booleano en cadena

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
3. Variables
4. Operador de asignación
5. Conversión de tipo
- 6. Entrada/Salida de datos básicos:** números enteros y reales y cadenas de caracteres
7. Formato de salida
8. Documentación de código
9. Indexación y funciones de cadenas o strings



6. Entrada/Salida de datos básicos

- Nuestros programas serían de muy poca utilidad si no fueran capaces de interactuar con el usuario
 - Para pedir información al usuario, se utiliza la función **input()**, así como los argumentos de línea de comandos
 - Para mostrar mensajes en pantalla, se utiliza la función **print()**

6. Entrada/Salida de datos básicos

- La función de salida **print()**:
 - muestra uno o más datos **formateados** por pantalla
- La función de entrada **input()**:
 - Espera a que el usuario escriba un texto y pulse return
 - Entonces prosigue la ejecución, devolviendo una cadena con el texto que tecleó el usuario
 - La función input admite un argumento opcional: una cadena con el mensaje que se le debe mostrar al usuario para este sepa qué información ha de introducir

6. Print imprime cadenas formateadas, no son las cadenas

```
>>> s = "hola\n a todos!"
>>>
>>> s
'hola\n a todos!'
>>> print(s)
hola
 a todos!
```

```
>>> s = 'This is Alvaro\'s text string'
>>> s
"This is Alvaro's text string"
>>> print(s)
This is Alvaro's text string
```

```
>>> s = 'este es un backslash \\'
>>> s
'este es un backslash \\'
>>> print(s)
este es un backslash \
>>>
```

Luego vemos mas cosas
sobre formatear cadenas

6. Entrada/Salida de datos básicos

Lee una cadena desde teclado y la muestra por pantalla:

```
x = input()
print(x)
```

Hola

Hola

El usuario escribe el texto que quiere en esta caja. Al pulsar retorno de carro, la cadena de texto escrita se guarda en la variable x. A continuación, el valor de x se muestra por pantalla gracias a la instrucción `print(x)`

En el ejemplo anterior, el usuario no sabe qué información ha de escribir. Incluyendo un mensaje como argumento del `input()`, el programa queda más claro:

```
edad = input('Dime tu edad:')
print('Tienes', edad, 'años')
```

Dime tu edad: 20

Tienes 20 años

El mensaje 'Dime tu edad:' se muestra al solicitar la información al usuario. La instrucción `print` escribe tres datos (separados por comas). Estos tres valores aparecen por pantalla separados por un espacio

6. Entrada/Salida de datos básicos

- La función `input()` devuelve una cadena de texto. En el caso de que necesitemos un número, tendremos que hacer una **conversión de tipo** a `int` o `float`
- Ejemplo: el siguiente programa indica si el año introducido por el usuario es o no bisiesto

```
year = int(input('Escribe un año:'))
esBisiesto = (year % 4 == 0)
esBisiesto = esBisiesto and ((year % 100) != 0)
esBisiesto = esBisiesto or ((year % 400) == 0)
print(esBisiesto)
```

Conversión
del texto
leído a un
número
entero

Escribe un año:

True

Un año es bisiesto si es divisible entre 4, excepto aquellos divisibles entre 100 pero no entre 400.

ejercicios 5 hasta 10

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
3. Variables
4. Operador de asignación
5. Conversión de tipo
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres

7. Formato de salida

8. Documentación de código
9. Indexación y funciones de cadenas o strings



7. Formato de salida: argumentos

```
print('Unamuno', 'Miguel')
```

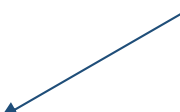
Unamuno Miguel

- La función `print()` usa un espacio en blanco como separador y un retorno de carro, `'\n'`, como carácter final
 - El argumento `sep` permite modificar el separador
 - El argumento `end` permite modificar el texto que se muestra al final


```
print('Unamuno', 'Miguel', sep=', ', end='.\n')
```

Unamuno, Miguel.

Añadimos un punto y un retorno de carro al final



Separamos las dos cadenas mediante una coma y un espacio

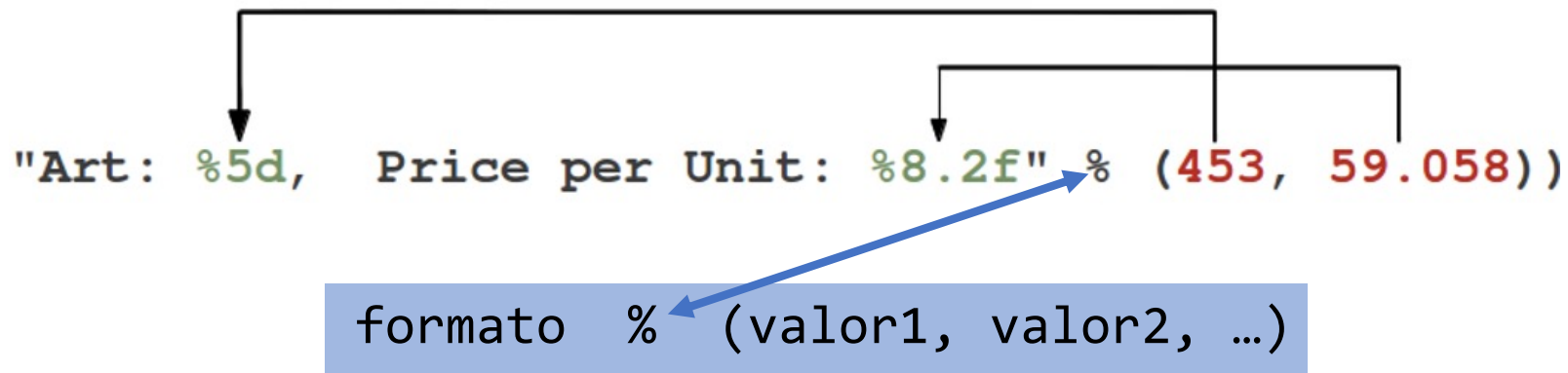


7. Formato de salida: 2 maneras

1. cadenas formateadas con %
2. cadenas formateadas con **str.format()**

7.1 Formato de salida: cadenas formateadas con %

- El operador especial % permite formatear la salida con gran flexibilidad.
- Se utiliza de la siguiente manera (muy similar a C, Perl, Bash por ejemplo)



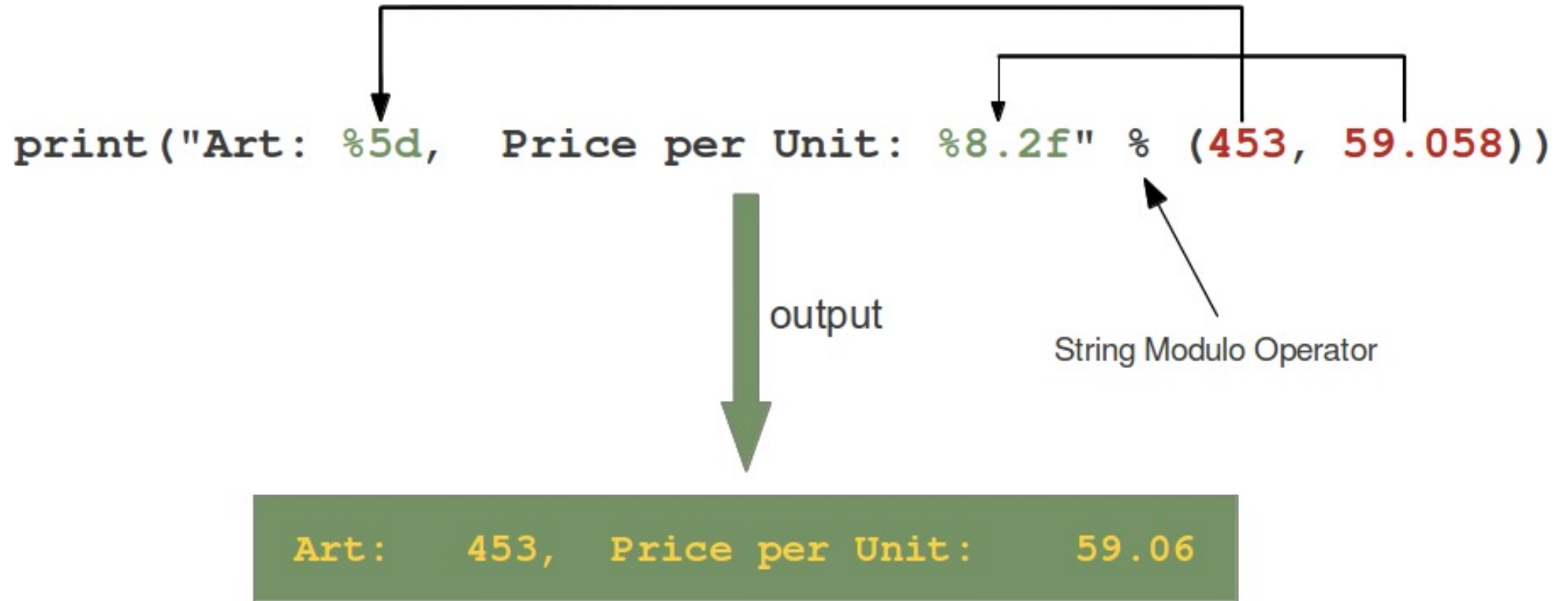
- formato es una cadena de texto que contiene dos clases de elementos:
 - Caracteres normales, que se imprimirán por pantalla directamente
 - **Especificadores de formato**, que definen el modo en que se visualizarán los valores que hay a continuación. Estos especificadores empiezan siempre con el carácter %
 - En el ejemplo hay 2 especificadores de formato:
 - %5d
 - %8.2f

7.1 Formato de salida: especificadores de formato

Los **especificadores de formato** tienen la siguiente sintaxis:

%[flags][width][.precision]type

- **type** : indica el tipo de dato que se va a mostrar (este dato es obligatorio, el resto son opcionales)
 - **%d**, para mostrar un número entero
 - **%f**, para mostrar un número real
 - **%s**, para mostrar una cadena de texto
- **.precision** : valor entero que indica el número de dígitos decimales a mostrar
- **width** : el tamaño mínimo. Cuando el tamaño del argumento es menor al mínimo, se completar los caracteres que falten rellenando con espacios
- **flags**:
 - **0** : completar los caracteres que falten rellenando con ceros en lugar de con espacios
 - **+** : Incluir siempre el signo (+ ó -)
 - **-** : Alinear a la izquierda (en lugar de a la derecha, por defecto)



Los especificadores de formato son

"%5d" and "%8.2f".

7.1 Formato de salida

"%06.2f"

```
>>> print("Price: $%06.2f" % (23.789))
Price: $023.79

>>> print("Price: $%06.2f" % (0.039))
Price: $000.04

>>> print("Price: $%06.2f" % (199.8))
Price: $199.80

>>> print("Price: $%06.2f" % (23))
Price: $023.00

>>> print("Price: $%06.2f" % (2324.17))
Price: $2324.17

>>> print("Price: $%06.2f" % (2324.17654567))
Price: $2324.18
```

7.1 Formato de salida: cadenas formateadas con %


- Ejemplos (suponemos que $a=7$, $b=2$ y $\text{nombre}=\text{'Miguel'}$)

Instrucción	Resultado
<code>print('Hola %s' % 'Miguel')</code>	Hola Miguel
<code>print('Tengo %d años' % 20)</code>	Tengo 20 años
<code>print('pi=%.2f' % 3.14159)</code>	pi=3.14
<code>print('%+d' % a)</code>	+7
<code>print('%03d' % a)</code>	007
<code>print('(%10d)' % a)</code>	(7)
<code>print('(%-10d)' % a)</code>	(7)
<code>print('%d * %d = %d' % (a, b, a*b))</code>	7 * 2 = 14
<code>print('%d / %d = %.2f' % (a, b, a/b))</code>	7 / 2 = 3.50
<code>print('%s tiene %d hijos' % (nombre, a))</code>	Miguel tiene 7 hijos
<code>print('\t... y Luis tiene %d' % b)</code>	... y Luis tiene 2
<code>print('%8s, %8s' % ('Unamuno', nombre))</code>	Unamuno, Miguel


7.2 Formato de salida:

cadenas formateadas con `str.format()`

- Formateo con % ya se llama OLD style
- New style es con la función `format()`



```
>> "El número {0} ha sido interpolado".format(1.23)
El número 1.23 ha sido interpolados
```



```
>> "Los números {0} y {1} han sido interpolado".format(1.23,9.9999)
Los números 1.23 y 9.9999 han sido interpolados
```

- {0} y {1} son **marcas**

7.2 Formato de salida

{[posicion]:[flags][width][.precision][codigo de tipo]}

- **posicion**: el argumento que hay que sustituir
 - 0 se ha sustituido por el primer argumento y la marca 1 por el segundo, etc
- **flags**: por ejemplo para alineación al a izquierda (<), a la derecha (>), en el centro (=), etc
- **width**: el tamaño mínimo. Cuando el tamaño del argumento es menor al mínimo, se completar los caracteres que falten rellenando con espacios
- **.precision**: numero de decimales con que queremos representar un numero flotante.
- **codigo de tipo**: caracter que indica el tipo de representacion que se desea. Es diferente segun el tipo de datos del valor.

```
>>> print('Un {0}, float: {1:3.2f} y decimal {2:d}'.format('Ejemplo', 12.5678, 20))  
Un Ejemplo, float: 12.57 y decimal 20
```

7.2 Formato de salida: diferentes codigos de tipo

numeros enteros:

- **b** : en binario
- **c**: como carácter Unicode.
- **d**: en base diez (es el valor por defecto).
- **o**: en octal.
- **x**: en hexadecimal.
- **n**: igual que d , pero como numero adaptado a la cultura local
 - por ejemplo el numero 100000
 - en Español, es 100.00 con punto
 - en América es 100,00 con coma (,)

7.3 Formato de salida: diferentes codigos de tipo

numeros flotantes:

- **g**: notacion en formato general
 - es exponente para numeros grandes
 - El valor por defecto si no hay codigo de tipo
- **e**: notacion exponente o notacion cientifica
- **f**: notacion de punto fijo
- **n**: igual que g, pero como numero adaptado a la cultura local.
 - Por ejemplo en España el punto decimal (2,54) se muestra como una coma, en América con (2.54)
- **%**: muestra el numero multiplicado por 100, en formato f y seguido de un símbolo de porcentaje.

Ejemplos

```
>>> txt = "Alineación a la izquierda {0:<8} del argumento, width 8.".format(49)
print(txt)
```

Alineación a la izquierda 49 del argumento, width 8.

```
>>> txt = "Alineación a la derecha {0:>7} del argumento, width 7.".format(49)
print(txt)
```

Alineación a la derecha 49 del argumento, width 7.

```
>>> txt = "Alineación en el centro {0:^6} del argumento, width 6.".format(49)
print(txt)
```

Alineación en el centro 49 del argumento, width 6.

```
>>> txt = "Alineación a la izquierda {0:a<9} del argumento, width 9, fill con a.".format(49)
print(txt)
```

Alineación a la izquierda 49aaaaaaaa del argumento, width 9, fill con a.

```
>>> txt = "Alineación a la derecha {0:*>8} del argumento, width 8, fill con *.".format(49)
print(txt)
```

Alineación a la derecha *****49 del argumento, width 8, fill con *.

```
>>> txt = "Alineación en el centro {0:-^10} del argumento, width 10, fill con -.".format(49)
print(txt)
```

Alineación en el centro ----49---- del argumento, width 10, fill con -.

Prueba diferentes codigos de tipo

- Fichero [try_formatting.py](#)
- En poliformat

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
3. Variables
4. Operador de asignación
5. Conversión de tipo
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres
7. Formato de salida
- 8. Documentación de código**
9. Indexación y funciones de cadenas o strings



8. Documentación de código

- Documentar el código de un programa es añadir suficiente información (**comentarios**) como para explicar lo que hace
- Para que también los humanos entiendan qué están haciendo y por qué.
- Documentar un programa es una necesidad que sólo se aprecia en su debida magnitud cuando:
 - hay errores que reparar
 - hay que extender el programa con nuevas capacidades
 - hay que adaptarlo a un nuevo escenario.
- Hay dos reglas que no se deben olvidar nunca:
 - todos los programas tienen errores y descubrirlos sólo es cuestión de tiempo si el programa se utiliza frecuentemente
 - todos los programas sufren modificaciones a lo largo de su vida, al menos todos aquellos que tienen éxito

es importante que el programa se entienda: para poder repararlo y modificarlo (por el programador original, bien por otro programador que le sustituya)

8. Documentación de código

- Los **comentarios** sirven para explicar a las personas que puedan leer el programa en el futuro, qué es lo que hace el programa, así como explicar algunas partes del código
 - Los comentarios se ignoran al ejecutar el código
- En Python, los comentarios se pueden poner de dos formas:
 - Comentarios de una sola línea:
 - se escribe el símbolo almohadilla (#) al comienzo del comentario. El comentario termina al final de la línea
 - Comentarios de varias líneas:
 - el comentario comienza y termina con unas comillas dobles triples (""")

8. Documentación de código: tipos

- Introducción
- Diseño
- Interpretación

8. Documentación de código: introducción

- Objetivo del programa
 - Entradas
 - Salidas
 - Restricciones (si hay)

```
#####  
# Este programa aplica el algoritmo para multiplicacion  
# a la Rusa.  
#  
# Entrada del programa:  
#     el multiplicando y el multiplicador  
# Salida del programa:  
#     el producto del multiplicando y el multiplicador  
# Restricciones:  
#     no funciona cuando el multiplicando es negativo  
#  
# author: profesores de la asignatura de programacion  
# date: 2019/2020  
# version: 2  
#####  
  
multiplicando = int(input("Cual es el multiplicando "))  
multiplicador = int(input("Cual es el multiplicador "))  
producto = 0  
|  
while multiplicando != 0:  
    if multiplicando % 2 != 0:  
        producto = producto + multiplicador  
    multiplicando = multiplicando//2  
    multiplicador = multiplicador * 2  
  
print ("El resultado es ", str(producto))
```

8. Documentación de código: diseño

- Indicar el esquema del diseño en el código

```
## multiplicar sucesivamente por 2 el multiplicando
## y dividir por 2 el multiplicador hasta que el
## multiplicador tome el valor 0. Mientras suma todos
## los multiplicandos correspondientes a los multiplicadores
## impares para obtener el producto.
|
while multiplicando != 0:
    if multiplicando % 2 != 0:
        producto = producto + multiplicador
    multiplicando = multiplicando//2
    multiplicador = multiplicador * 2

print ("El resultado es ", str(producto))
```

8. Documentación de código: interpretación

- A veces ayuda al lector a explicar expresiones en términos del mundo del problema.
- Preferimos escribir este comentario en la misma línea:

```
while multiplicando != 0:  
    if multiplicando % 2 != 0: #solo funciona para multiplicando positivo  
        producto = producto + multiplicador  
    multiplicando = multiplicando//2  
    multiplicador = multiplicador * 2
```

Índice

TEMA 2

1. Tipos de datos: numéricos, boolean, cadenas
2. Expresiones y operadores (aritméticos, relacionales, lógicos)
3. Variables
4. Operador de asignación
5. Conversión de tipo
6. Entrada/Salida de datos básicos: números enteros y reales y cadenas de caracteres
7. Formato de salida
8. Documentación de código
- 9. Indexación y funciones de cadenas o strings**



9. Indexación de cadenas

- Podemos acceder a cada uno de los caracteres de una cadena utilizando el operador de indexación []
 - El índice del elemento al que queremos acceder debe encerrarse entre corchetes
- Si *a* es una cadena, *a*[0] es el primer carácter de la cadena, *a*[1] el segundo, y así sucesivamente
 - Los índices de la cadena 'Hola, mundo.' se muestran en esta figura:

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.

```
a = 'Hola, mundo.'  
print(a[0])  
print(a[3])
```

H
a

9. Indexación de cadenas

```
>>> a = "Hola, mundo."
```

```
>>> a[0]
```

```
'H'
```

```
>>> a[1]
```

```
'o'
```

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.

- El último carácter de una cadena `a` es `a[len(a)-1]`

```
>>> a[len(a)-1]
```

```
'.'
```

- Es muy importante que el índice que usemos sea válido:
- Si usamos un índice fuera de rango, Python lanza una excepción *IndexError*

```
>>> a[len(a)]
```

```
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
IndexError: string index out of range
```


9. Indexación de cadenas

- También se pueden utilizar índices negativos: los valores negativos acceden a los caracteres de derecha a izquierda. El último carácter de una cadena tiene índice -1 , el penúltimo, -2 , y así sucesivamente

```
>>> a = "Ejemplo"
```

```
>>> a[-1]
```

```
'o'
```

```
>>> a[-2]
```

```
'l'
```

```
>>> a[-4]
```

```
'm'
```

```
>>> a[-8]
```

```
Traceback (most recent call last):
```

```
File "<pyshell>", line 1, in <module>
```

```
IndexError: string index out of range
```

```
>>> |
```

0	1	2	3	4	5	6
E	j	e	m	p	l	o
-7	-6	-5	-4	-3	-2	-1

9. Subcadenas: el operador de corte

- El operador de corte se denota con dos índices separados por dos puntos (:), dentro de los corchetes de indexación
 - La expresión `a[i:j]` devuelve la subcadena formada por los caracteres desde `a[i]` hasta `a[j-1]`

```
>>> a = "Ejemplo"
>>> a[2:5]
'emp'
>>> a[0:4]
'Ejem'
.
```

0	1	2	3	4	5	6
E	j	e	m	p	l	o
-7	-6	-5	-4	-3	-2	-1

9. Subcadenas: el operador de corte

- Podemos dejar alguno (o ambos) de los índices vacíos si queremos comenzar por el primer carácter o terminar por el último:

```
>>> a[:4]  
'Ejem'
```

```
>>> a[2:]  
'emplo'
```

```
>>> a[:-4]  
'Eje'
```

```
>>> a[-2:]  
'lo'
```

0	1	2	3	4	5	6
E	j	e	m	p	l	o
-7	-6	-5	-4	-3	-2	-1

9. Subcadenas: el operador de corte

- Al igual que en los rangos, al operador de corte se le puede añadir un tercer parámetro que indica el **incremento** del índice y la **dirección**

- Un 2 indica ir
 - saltando los caracteres de 2 en 2
 - de la izquierda hacia la derecha
- Un -1 indica ir
 - visitando los caracteres uno por uno
 - de derecha a la izquierda

0	1	2	3	4	5	6
E	j	e	m	p	l	o
-7	-6	-5	-4	-3	-2	-1

```
>>> a = "Ejemplo"
>>> a[::2]
'Eepo'

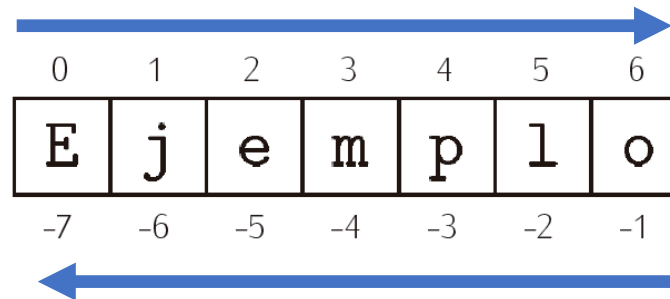
>>> a[::-1]
'olpmejE'

>>> a[1:4:2]
'jm'

>>> a[1::-1]
'jE'
```

9. Subcadenas: el operador de corte

- Un numero POSITIVO indica ir de izquierda a la derecha
- Un numero NEGATIVO indica ir de derecha a la izquierda



```
>>> a[-4:-2:1]
'mp'
>>> a[-4:-2:-1]
''
```

```
>>> a[-2:-4:-1]
'lp'
>>> a[-2:-4:1]
''
```

```
>>> a[1:3:1]
'je'
>>> a[1:3:-1]
''
```

```
>>> a[3:1:1]
''
>>> a[3:1:-1]
'me'
```

9. Métodos de manejo de cadenas

Método	Descripción	Ejemplo (si <code>s = 'Valencia'</code>)
<code>s.count(t)</code>	Número de ocurrencias de la cadena <code>t</code> dentro de <code>s</code>	<pre>>>> s.count('a') 2</pre>
<code>s.find(t)</code>	Posición de la 1ª ocurrencia de <code>t</code> en <code>s</code> . Si no esta devuelve -1	<pre>>>> s.find('en') 3 >>> s.find('Pa') -1</pre>
<code>s.find(t, b, e)</code>	Posición de la 1ª ocurrencia de <code>t</code> en <code>s[b:e]</code>	<pre>>>> s.find('en',5,8) -1 >>> s.find('en',0,3) -1</pre>
<code>s.upper()</code>	Devuelve una copia de <code>s</code> en mayúsculas	<pre>>>> s.upper() 'VALENCIA'</pre>
<code>s.lower()</code>	Devuelve una copia de <code>s</code> en minúsculas	<pre>>>> s.lower() 'valencia'</pre>
<code>s.startswith(t)</code>	Indica si <code>s</code> comienza por <code>t</code>	<pre>>>> s.startswith('Val') True</pre>
<code>s.endswith(t)</code>	Indica si <code>s</code> termina con <code>t</code>	<pre>>>> s.endswith('ia') True</pre>
<code>s.replace(u, v)</code>	Devuelve una copia de <code>s</code> con todas las apariciones de <code>u</code> reemplazadas por <code>v</code>	<pre>>>> s.replace('a', 'oo') 'Voolencioo'</pre>
<code>s.isdigit()</code>	Devuelve True si <code>s</code> consiste solo de dígitos 0 a 9	<pre>>>> 'Hola!'.isdigit() False >>> '123'.isdigit() True</pre>

¡Solo hay una forma de aprender a programar!



practicando, practicando, practicando, practicando