Informatica

Sesion 1 Teoria

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ferran Juan Baruel
Ciències de la Computació / 2017

Temas a tratar en la Sesión 1

- Tipos de datos
- Variables
- Operadores y precedencia
- Algebra de Boole y expresiones lógicas
- Funciones

Sesión 1

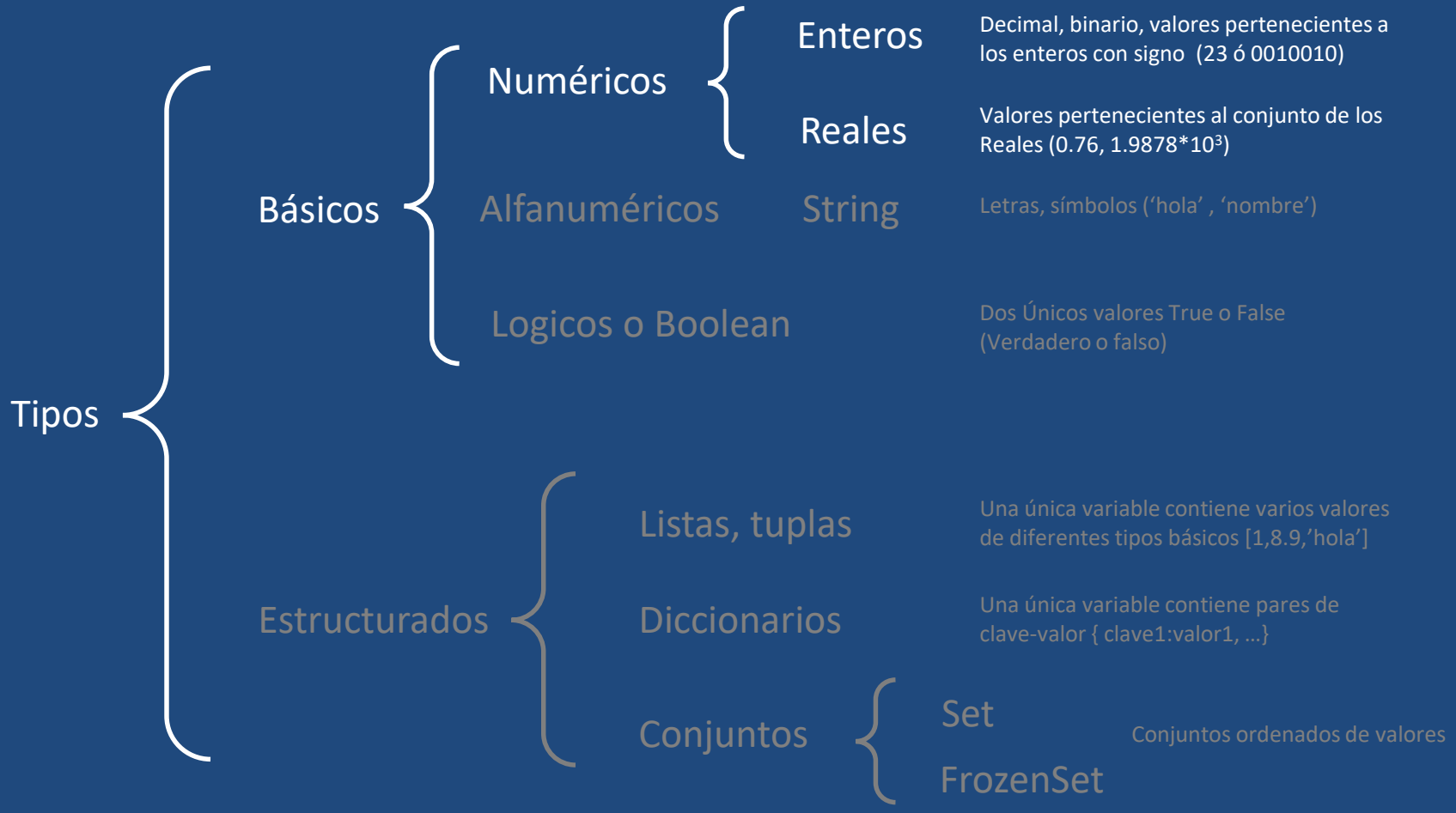
- Tipos de datos
- Variables
- Operadores y precedencia
- Algebra de Boole y expresiones lógicas
- Funciones

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ferran Juan Baruel
Ciències de la Computació / 2017

Tipos de datos

Tipos de datos



Tipos numéricos:
Codificación, Rango y precisión

- Tipo entero o valor que pertenece a los enteros $(x \in \mathbb{Z})$:
  - El tipo entero de Python está codificado en binario natural y en complemento a 2 [F.Virgos]. Almacena números de cualquier rango y precisión limitado por la memoria disponible en la máquina.
- Tipo Float o que pertenece al conjunto de los Reales $(x \in \mathbb{R})$:
  - El tipo Float se almacena utilizando el tipo Double de C de 64 bits y codificación estándar IEEE 754 : 1 bit para el signo, 11 para el exponente y 52 para la mantisa. (ver https://en.wikipedia.org/wiki/IEEE_floating_point#IEEE_754-2008)
  - Ejemplo: -2.98873634541324356e4 es $2.98873634541324356 * 10^4 \rightarrow$ 29887.3634541324356
  - El valor más grande que puede representar es el 1.7976931348623157e+308 y el mínimo más cercano al cero es el 2.2250738585072014e-308 (tanto positivos como negativos)
  - Los Número Reales tienen una precisión de 17 dígitos

[1] Ferran Virgos Codificacion

Tipos numéricos:
Codificación, Rango y precisión

```
>>> 2 * 9
18
>>> 0x33 + 12
63
>>> 0b1001
9
>>> 0o22
18
>>> 2**9
512
>>> 2**1024
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
```

```
>>> 0.54 * 7
3.7800000000000002
>>> 0.73 *9.87
7.205099999999999
>>> 23.09 ** 6
151545692.81779075
>>> 1.2345678909876543212345678 9 + 1.0
2.2345678909876545
```

Tipos de datos

El tipo cadena

- Una cadena es una secuencia de caracteres (letras, números, espacios, marcas de puntuación, etc.) y en Python se distingue porque va encerrada entre comillas simples o dobles. Por ejemplo, 'cadena' , 'otro ejemplo' , "1, 2 1 o 3" , '!Si!' , "...Python" son cadenas.
- Las cadenas pueden usarse para representar información textual: nombres de personas, nombres de colores, etc. También pueden ser almacenadas en variables.
- La codificación de las cadenas sigue el estándar ASCII en las que cada carácter está codificado en 8 bits.

```
>>> hola buenos dias
SyntaxError: invalid syntax
>>> 'hola buenos dias'
'hola buenos dias'
```

```
>>> nombre = pepe
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    nombre = pepe
NameError: name 'pepe' is not defined
>>> nombre = 'pepe'
>>> nombre
'pepe'
```

Estándares de codificación

- ASCII : Codificación estándar de 8 bits que permite codificar caracteres especiales para los diferentes países que comparten el mismo alfabeto. Una variante estandarizada para nuestro país es la tabla ISO-8859-15
- UNICODE: Estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y diferentes alfabetos (*)

* https://unicode-table.com/es/

Tabla ASCII

- Los 127 primeros bits es común para todos los idiomas occidentales. El resto son específicos de cada idioma.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		▯		▯	▯	▯	▯	▯	▯			▯	□		▯	▯
16	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯				
32	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	▯
128	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
144		‘	’	“	”	•	–	—		™	š	›	œ		ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Funciones de información

- La función “ord(caracter)” indica el código ASCII numérico que corresponde al carácter seleccionado.
- La función “chr(valor)” muestra el carácter que está asociado al valor ASCII indicado.

```
>>> ord('a')
97
>>> ord('ñ')
241
>>> chr(254)
'þ'
>>> chr(64)
'@'
```

Tipos de datos

El tipo Boolean

- El tipo Boolean es un tipo de datos que sólo contiene dos valores : **True** y **False**.
- El tipo de datos se puede asociar a una variable para ser posteriormente analizada para la toma de decisiones.
- El tipo de datos también es el resultado de operaciones lógicas con valores de diferentes tipos
- Su uso está dedicado a la toma de decisiones en los diferentes algoritmos y bloques de decisión.

```
>>> 2 > 6
False
>>> b = False
>>> b
False
>>> 9 == 8
False
>>> 6 > 2
True
```

Sesión 1

- Tipos de datos
- Variables
- Operadores y precedencia
- Algebra de Boole y expresiones lógicas
- Funciones

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ferran Juan Baruel
Ciències de la Computació / 2017

Definición de variables en Python

- Los tipos de datos son asignados en el momento de asociar una variable a un valor
- Podemos ver el tipo de cada variable llamando a la función "type(var)"

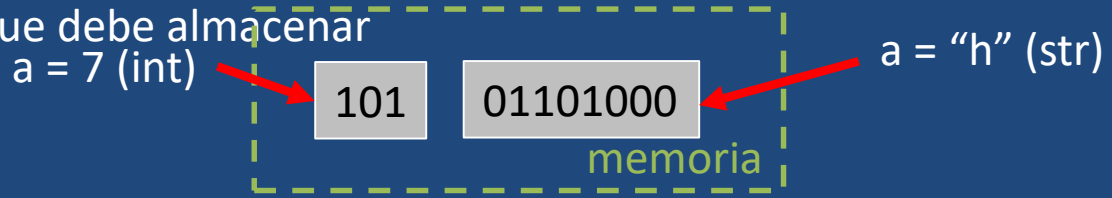
```
>>> a = 2
>>> type(a)
<class 'int'>
>>> b = 0.87
>>> type(b)
<class 'float'>
>>> c = 'hola'
>>> type(c)
<class 'str'>
>>> d = True
>>> type(d)
<class 'bool'>
```

```
>>> a = 2
>>> type(a)
<class 'int'>
>>> a = 'hola'
>>> type(a)
<class 'str'>
```

Los datos pueden cambiar dinámicamente de tipo al asignar valores de tipos diferentes

Variables

- Una variable es la manera que tiene el ordenador de almacenar temporalmente un valor.
- Podemos intuir que los valores de las variables se almacenan directamente en una posición de memoria. La variable es un apuntador a ésta posición de memoria (a modo de indicación de su localización)
- Los valores se almacenan siempre en binario y en las diferentes codificaciones dependiendo del tipo de valor que está almacenado
- Una variable necesita un nombre y un valor.
  - Por ejemplo, la variable entera a = 23 tiene el nombre a y el valor 23
- El acto de dar valor a una variable se denomina asignación. Al asignar un valor a una variable que no existía, Python reserva un espacio en la memoria, almacena el valor en él y crea una asociación entre el nombre de la variable y la dirección de memoria de dicho espacio. Python reconoce en el momento de la asignación el tipo de dato que debe almacenar



Variables. Asignación

- La asignación de un valor a una variable se realiza con la sentencia :

variable = expresión

- El orden es importante. “expresión = variable” no es correcto
- Una asignación no es una ecuación matemática, sino una acción consistente en :
  - 1. evaluar la expresión a la derecha del símbolo igual (=), y
  - 2. guardar el valor resultante en la variable indicada a la izquierda del símbolo igual.
- Se puede asignar valor a una misma variable cuantas veces se quiera. El efecto es que la variable, en cada instante, sólo *recuerda* el último valor asignado. . . hasta que se le asigne otro.
- Una expresión puede ser un valor o una expresión que tenga como resultado un valor.

Variables. Nombres

- El nombre de una variable es su identificador.
- Hay unas reglas precisas para construir identificadores.
  - Estar formado por letras minúsculas, mayúsculas, dígitos y/o el carácter de subrayado (_)
  - El primer carácter nunca ha de ser un dígito.
  - No se recomiendan letras o símbolos impropios del alfabeto inglés, (á, é … , ñ, ç .etc).
  - Palabras reservadas (por ser utilizadas por el lenguaje):

and, assert, break, class, continue, def, del, elif,
else, except, exec, finally, for, from, global, if,
import, in, is, lambda, none, not, or, pass, print,
raise, return, try, while y yield.

Variables. Nombres

- Identificadores válidos:
- h, x, Z, velocidad, aceleracion, fuerza1, masa _ 2, _ a, a _ , prueba _ 123, desviacion _ tipica.
- Diferencia entre mayúsculas y minúsculas: area, Area y AREA son tres variables diferentes.
- No es válido el espacio en blanco .Por ejemplo, edad media. En cambio, sí es válido edad_media o EdadMedia
- Los caracteres especiales (ñ, ç, y/o vocales acentuadas) son válidas aunque no recomendables pues su exportación a otros sistemas con diferente codificación harían el programa inservible. Se recomienda utilizar el alfabeto anglosajón (UK o USA)

Asignar nombres a variables

- Recomendación:
- El nombre de las variables debe tener sentido dentro del ámbito o contexto donde se utilizan. De esta manera son fáciles de recordar y ayuda a relacionar y localizar los valores con más facilidad.
- Las variables han de guardar relación con los datos del problema. Si vas a utilizar una variable para almacenar una distancia, llama a la variable *distancia* y evita nombres que no signifiquen nada; de este modo, los programas serán más legibles.
- Por ejemplo, variables con el nombre i, j, k, cont, se usan frecuentemente como contadores, x, y, z como valores reales, etc.

Variables

- >>> pi =3.141592653589793
- >>> radio = 1.5427326
- >>> area = pi * radio ** 2
- >>> perimetro = 2 * pi * radio
- >>> area
- 7.477065521391141
- >>> perimetro
- 9.693274805226961
- >>> x = 2
- >>> x
- 2
- >>> x = x + 1
- >>> x
- 3

Sesión 1

- Tipos de datos
- Variables
- Operadores y precedencia
- Algebra de Boole y expresiones lógicas
- Funciones

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ferran Juan Baruel
Ciències de la Computació / 2017

Operadores aritméticos

- Los operadores aritméticos son la suma, resta , multiplicación y división (con sus variantes de división Real y entera), y el operador de potencia.
- La división la podemos definir de tres maneras diferentes en función del tipo de división que se necesite. Así con una barra "/" realizamos una división Real, en el que el resultado es un número de tipo Real o 'float'. Con dos barras "//" se calcula sólo el cociente de la división, y con el símbolo '%' se calcula el residuo de la división, dando ambos resultados números enteros.
- >>>7 / 4 → 1.75 → Numero Real
- >>>7 // 4 → 1 → Numero entero (Cociente)
- >>>7 % 4 → 3 → Numero entero (Resto)

```
>>> 2 + 3
5
>>> 8 - 4
4
>>> 9 * 6
54
>>> 7 / 4
1.75
>>> 7 // 4
1
>>> 7 % 4
3
>>> 8 ** 3
512
```

Precedencia de operadores

- La precedencia de los operadores aritméticos son:

Operación	Operador	Asociatividad	Precedencia
Paréntesis	()	Interior a exterior	0
Exponenciación	**	Por la derecha	1
Identidad	+	--	2
Cambio de signo	-	--	2
Multiplicación	*	Por la Izquierda	3
División	/	Por la Izquierda	3
Módulo	%	Por la Izquierda	3
Suma	+	Por la Izquierda	4
Resta	-	Por la Izquierda	4

Precedencia de operadores

- Ejemplo:

  4*5 – 3 + 8**2 / 4*3

  4*5 – 3 + 64 / 4*3

  20 – 3 + 64 / 4*3

  20 – 3 + 16 * 3

  20 – 3 + 48

  17 + 48 → 65

Precedencia de operadores

- Uso de paréntesis: Modifican el orden de operación
- Los paréntesis están en la primera posición de precedencia. Se ejecutan desde el más interno al más externo.

  (4 * (5-3) + 8) ** 2 / (4 * 3)

  (4*2+8) ** 2/ (4*3)

  (8+8) **2/ (4*3)

  16 **2/ (4*3)

  16**2 / 12

  256/12

  21.3333333333333332

Precedencia de operadores

- Para el cálculo de raíces se recomienda (y en algunos casos se obliga por precisión y para el cálculo del valor más preciso) a poner el valor de la potencia como una fracción.

  $\sqrt[3]{64}$ → 64 **(1/3) → 4.0

- Se debe poner la potencia entre paréntesis para que evalúe primero la división y luego la potencia. Si no pusiéramos el paréntesis el resultado seria diferente, ya que por precedencia, primero elevaría el valor 64 a la potencia 1 y luego dividiría el resultado por 3
- >>> 64 ** 1 / 3
- >>> 64 / 3
- >>> 21.333333333333332

Operadores Lógicos

- OPERADORES RELACIONALES (DE COMPARACIÓN)
- Nos sirven para evaluar una condición entre dos valores

Símbolo	Significado
==	Igual que
!=	Distinto que
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

- OPERADORES LOGICOS
- Nos sirven para evaluar más de una condición simultáneamente.

Operador	Ejemplo
and (y)	5 == 7 and 7 < 12
	a < b and c > 7
or(o)	12 == 12 or 15 < 7

Precedencia de operadores

- La precedencia de los operadores Lógicos van a continuación de los aritméticos:

Operación	Operador	Asociatividad	Precedencia
Igual que	==	---	5
Distinto de	=!	---	5
Menor que	<	---	5
Menor o igual que	<=	---	5
Mayor que	>	---	5
Mayor o Igual que	>=	---	5
Negación	not	(*)	6
Intersección	in	(*)	6
Conjunción	and	Por la izquierda	7
Disyunción	or	Por la izquierda	8

(*) estos dos operadores solo se encuentran juntos en la combinación “not in” para indicar que un elemento NO se encuentra dentro de otro

Precedencia de operadores

- La paradoja de la Asociatividad de los comparadores: Python puede evaluar expresiones como “2 < 1 < 4” y más complejas , expresiones que en otros lenguajes no están permitidas.

Lenguaje C	Pascal	Python
2<1<4	2<1<4	2<1<4
Evaluación por la izquierda: 2<1 → 0(falso) Y 0<4 → 1(cierto)	Evaluación por la izquierda: 2<1 → False Y False<4 → Error	Lo evalúa como 2<1 and 1<4 → False and True → False
La evaluación del resultado no es correcta. No se recomienda su uso.	Más estricto, no permite operaciones con valores de diferentes tipos (boolean y integer)	La evaluación del resultado es correcta.

Operadores de cadenas

- Conactenación o suma: Se utiliza el símbolo ‘+’
- Repetición : se usa el operador “*” para la acción de repetir el texto

```
>>> a = 'Hola'
>>> b = 'Alberto'
>>> c = 37
>>> texto = a + ', '+ b + ': tenemos ' + str(c) + ' piezas almacenadas'
>>> texto
'Hola, Alberto: tenemos 37 piezas almacenadas'
>>> b = '******************************'
>>> c = '*'*30
>>> b
'******************************'
>>> c
'******************************'
>>> d = a*3
>>> d
'HolaHolaHola'
```

Comparación de cadenas

- La comparación de cadenas es directa y no necesita de pasos intermedios. Se compara por los códigos numéricos de cada letra.
- En el caso de los operadores == y != el significado está claro: dos cadenas son iguales si son iguales carácter a carácter, y distintas en caso contrario.
- Python distingue entre mayúsculas y minúsculas, ya que sus códigos son diferentes. (ver código ASCII en transparencias anteriores)

```
>>> a = 'Pedro'
>>> b = 'Manuel'
>>> a > b
True
```

P	e	d	r	o	
80	101	100	114	111	

M	a	n	u	e	l
77	97	110	117	101	108

Sesión 1

- Tipos de datos
- Variables
- Operadores y precedencia
- Algebra de Boole y expresiones lógicas
- Funciones

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ferran Juan Baruel
Ciències de la Computació / 2017

Expresiones Lógicas o Booleanas

- El termino Booleano proviene de George Boole, creador de la lógica de Boole o Algebra de Boole (*)
- El Álgebra de Boole también llamada álgebra booleana, es en informática una estructura algebraica que esquematiza las operaciones lógicas Y, O, NO (AND, OR, NOT) (**)

(*) (ver https://en.wikipedia.org/wiki/George_Boole)

(**)ver https://es.wikipedia.org/wiki/%C3%81lgebra_de_Boole)

Expresiones Lógicas o Booleanas

- Se trabajan con dos únicos valores:
  - Verdadero – Falso / True – False / Sí – No / 1 – 0
- Y tres operadores :
  - AND, OR, NOT
- Python incluye potros operadores como IN o IS
- Operador AND: El resultado será True siempre que ambos operandos sean True, en otro caso será False
- Operador OR: El Resultado será True si uno o ambos son True. Si ambos son False el resultado es False

A	B	A And B	A OR B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Evaluación de variables como lógicas

- Cualquier variable puede ser evaluada como una variable lógica o Booleana, para su uso en estrtucturas de decision
- Los siguientes valores pueden ser evaluados como False
  - None,
  - False
  - Un valor cero de cualquier tipo numérico: 0, 0.0, …
  - colecciones y secuencias vacías: '', (), [], {}, range(0)
  - Las operaciones y funciones que tienen siempre un resultado booleano devuelven 0 o **False** para falso y 1 o **True** para verdadero
  - En cualquier otro caso, la variable puede evaluarse como **True**

- Más información: https://docs.python.org/3/library/stdtypes.html#truth-value-testing

Expresiones Lógicas o Booleanas

- Operador Not: Cambia el estado del operando
- Operador In : El resultado es True si el operando de la izquierda se encuentra en el de la derecha

A	NOT A
True	False
False	True

```
>>> ciudad = 'Barcelona'
>>> patron = 'cel'
>>> patron in ciudad
True
>>> secuencia = 2,7,4,3,7,4,5,6,8
>>> valor = 4
>>> valor in secuencia
True
>>> 0 in secuencia
False
```

Propiedades y teoremas

- Conmutativa:
  - A OR B = B OR A
  - A AND B = B AND A
- Distributiva:
  - A AND (B OR C) = A AND B OR A AND C
- Asociativa:
  - A OR (B AND C) = (A OR B) AND (A OR C)
- A OR True = True
- A AND False = False
- A OR (A AND B) = A
- A AND (A OR B) = A
- Leyes de Morgan:
- NOT (A OR B) = NOT A AND NOT B
- NOT (A AND B) = NOT A OR NOT B

Expresiones Lógicas o Booleanas

- Ejemplos:
- x está entre 5.0 y 10.0 o entre 15.0 y 20.0.
- (x >= 5.0 and x <=10.0) or (x >= 15.0 and x <=20.0)
- a i b no son mas grandes que 5.
- a <=5 and b <=5
- x es un múltiple de 10 ubicado entre 3000 y 4000.
- x % 10 == 0 and x >= 3000 and x <= 4000
- x % 10 == 0 and (3000 <= x <= 4000)
- x es un número par ubicado entre 100 y 1000 que no es múltiple de 10.
- x % 2 == 0 and x >= 100 and x <= 1000 and x % 10 != 0
- x % 2 == 0 and (100 <= x <= 1000) and x % 10 != 0
- x es un número divisible por 8 y acabado en 4
- x % 8 == 0 and x % 10 == 4

Expresiones Lógicas o Booleanas

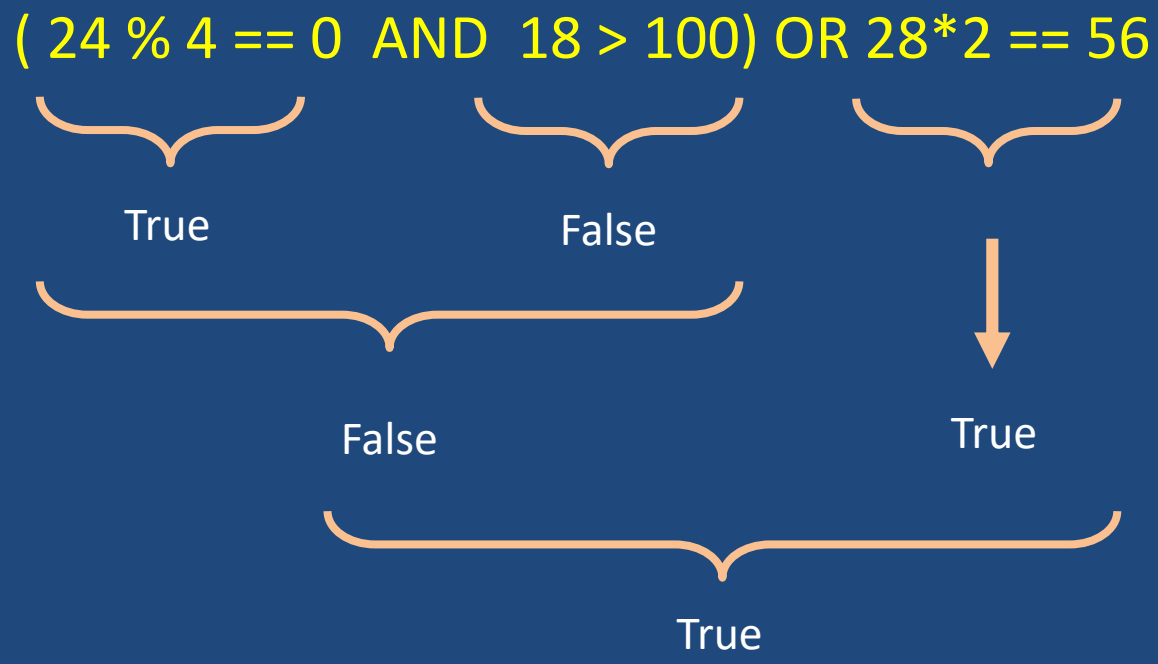
- Ejemplos:
- El valor de a corresponde a un año bisiesto:
  - Para que un año sea bisiesto debe cumplirse que el valor sea divisible por 4 , exceptuando las centenas que no lo son pero sí las centenas divisibles por 400

(a % 4 == 0 AND a % 100 != 0) OR a % 400 == 0

"a es divisible por 4" "a no es divisible por 100" "a es divisible por 400"

Cómo evaluar una expresión lógica

- Se evalúa por partes y siempre respetando la precedencia de los operadores
- Ejemplo:



Sesión 1

- Tipos de datos
- Variables
- Operadores y precedencia
- Algebra de Boole y expresiones lógicas
- Funciones

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ferran Juan Baruel
Ciències de la Computació / 2017

Funciones

- En programación, una función es una secuencia de instrucciones que forman una tarea determinada. La función realiza siempre una acción específica y está específicamente diseñada para ser reutilizada tantas veces como sea necesario.
- Python tiene una serie de funciones predefinidas (*):
  - Por ejemplo las funciones input(), print(), abs() o round()
- Más funciones están alojadas en diferentes módulos que deben ser invocados para poder hacer uso de ellas.
  - Por ejemplo, el módulo Math(**) contiene una colección de funciones matemáticas que nos pueden ahorrar mucho el trabajo de cálculo.
- Las funciones reducen la duplicación de código en un programa. Al tener una tarea específica atendida en un correcto bloque de código que podemos importar y llamar cuando queramos, no necesitamos duplicar su aplicación.

```
>>> x = -1
>>> y = abs(x)
>>> y
1
>>> import math
>>> x = 23
>>> y = math.sqrt(x)
>>> y
4.795831523312719
>>> z = round(y,3)
>>> print(' valor de z:',z)
 valor de z: 4.796
```

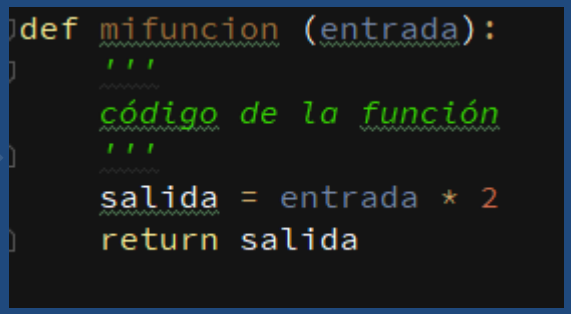
(*) https://docs.python.org/3/library/functions.html

(**) https://docs.python.org/3.6/library/math.html

Crear una función

- Una función se define con la palabra clave **def** seguido del nombre de la función , paréntesis y seguido de dos puntos:
- Entre los paréntesis podemos poner, opcionalmente, los parámetros de entrada
- Luego, y con una tabulación de 4 espacios, el código de la función.
- Una función puede o no puede devolver valores de salida. Si una función quiere devolver valores de salida, lo hace mediante la palabra clave **return**, seguida por la salida deseada. Si la función no tiene ninguna sentencia **return** en el cuerpo, es valor devuelto es **None**.

Entrada (opcional)



Salida (opcional)

Invocación y uso

- La función definida no es utilizada hasta que es llamada por el programa
- La función será invocada por su nombre y entre paréntesis se incluirán los valores que son necesarios para su ejecución. La única condición es que los valores que se envían a la función deben ser del mismo tipo que los parámetros de la función

```
def mifuncion (entrada):
    '''
    código de la función
    '''
    salida = entrada * 2
    return salida

# Programa #
print('el valor de a es 27')
a = 27
b = mifuncion(a)
print('el valor de b es ', b)
```

Al ejecutar:

```
el valor de a es 27
el valor de b es  54

Process finished with exit code 0
```

Parámetros de salida

- Una función puede retornar uno o más valores, o ninguno. Si se devuelven más de un valor, la función retorna una lista de valores que puede ser asignada a una variable o a tantas variables como elementos tenga la lista. Al utilizar la palabra clave **return**, se indican los valores a retornar separados por

```
File  Edit  Format  Run  Options  Window  Help
def datosfecha(fecha):
    d,m,a = fecha.split('/')
    return d,m,a
```

```
>>> datosfecha('10/10/2016')
('10', '10', '2016')
```

```
>>> dd,mm,aa = datosfecha('26/07/2015')
>>> print(dd)
26
>>> print(mm)
07
>>> print(aa)
2015
```

Ejemplos:

- Realizamos una función que recibe las coordenadas x,y de un punto, calcula y devuelve las coordenadas polares correspondientes. Luego creamos un programa main que realiza la llamada a la función.

```
def cart2polar (x,y):
    from math import atan2, sqrt
    m = sqrt(x**2 + y**2)
    a = atan2(y,x)
    return m,a

if __name__ == '__main__':
    from math import pi
    cx = float(input('coordenada x:'))
    cy = float(input('coordenada y:'))
    pm,pa = cart2polar(cx,cy)
    print('Las coordenadas (',cx,',',cy,'):')
    print('Módulo =',pm)
    print('Argumento (en grados):',pa*180/pi)
```

Al ejecutar:

```
coordenada x:4.58
coordenada y:7.69
Las coordenadas ( 4.58 , 7.69 ):
Módulo = 8.950558641783205
Argumento (en grados): 59.22287401149206
```