Computational Thinking with Programming

@cse_bennett @csebennett

Boolean Expressions

- The **Boolean data type** contains two Boolean values, denoted as **True** and **False** in Python.
- A **Boolean expression** is an expression that evaluates to a Boolean value.
- Boolean expressions are used to denote the conditions for selection and iterative control statements.

Relational Operators

- **Relational expressions** are a type of Boolean expression, since they evaluate to a Boolean result.
- These operators **not only apply** to numeric values, but to any set of values that has an ordering, such as **strings**.
- Note the use of the comparison operator == for determining if two values are equal. This, rather than the (single) equal sign = is used since the equal sign is used as the assignment operator.
- This is often a source of confusion for new programmers,
  - num = 10 variable num is assigned the value 10
  - num == 10 variable num is compared to the value 10

Relational Operators

Relational Operators	Example	Result
== equal	10 == 10	True
!= not equal	10 != 10	False
< less than	10 < 20	True
> greater than	'Alan' > 'Brenda'	False
<= less than or equal to	10 <= 10	True
>= greater than or equal to	'A' >= 'D'	False

Example

```
x=5
print(x>3)
print(x==5)
print(x>=5)
print(x<=5)
print(x<6)
print(x!=6)
```

True

True

True

True

True

True

Membership Operators

Membership Operators	Examples	Result
in	10 in (10, 20, 30)	True
	red in ('red','green','blue')	True
not in	10 not in (10, 20, 30)	False

Boolean Operators

- George Boole, in the mid-1800s, developed what we now call *Boolean algebra*.
- His goal was to develop an algebra based on **true/false** rather than numerical values.
- Boolean algebra contains a set of **Boolean** (**logical**) **operators** , denoted by **and**, **or**, and **not** in Python.

Boolean Operators

- Logical **and** is **true** only when *both* its operands are **true**—otherwise, it is false.
- Logical **or** is **true** when *either or both* of its operands are **true**, and thus **false** only when both operands are **false**.
- Logical **not** simply reverses truth values—**not False** equals **True**, and **not True** equals **False**.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Boolean Logic Truth Table

x	y		x and y	x or y	not x
False	False		False	False	True
True	False		False	True	False
False	True		False	True	
True	True		True	True	

Example

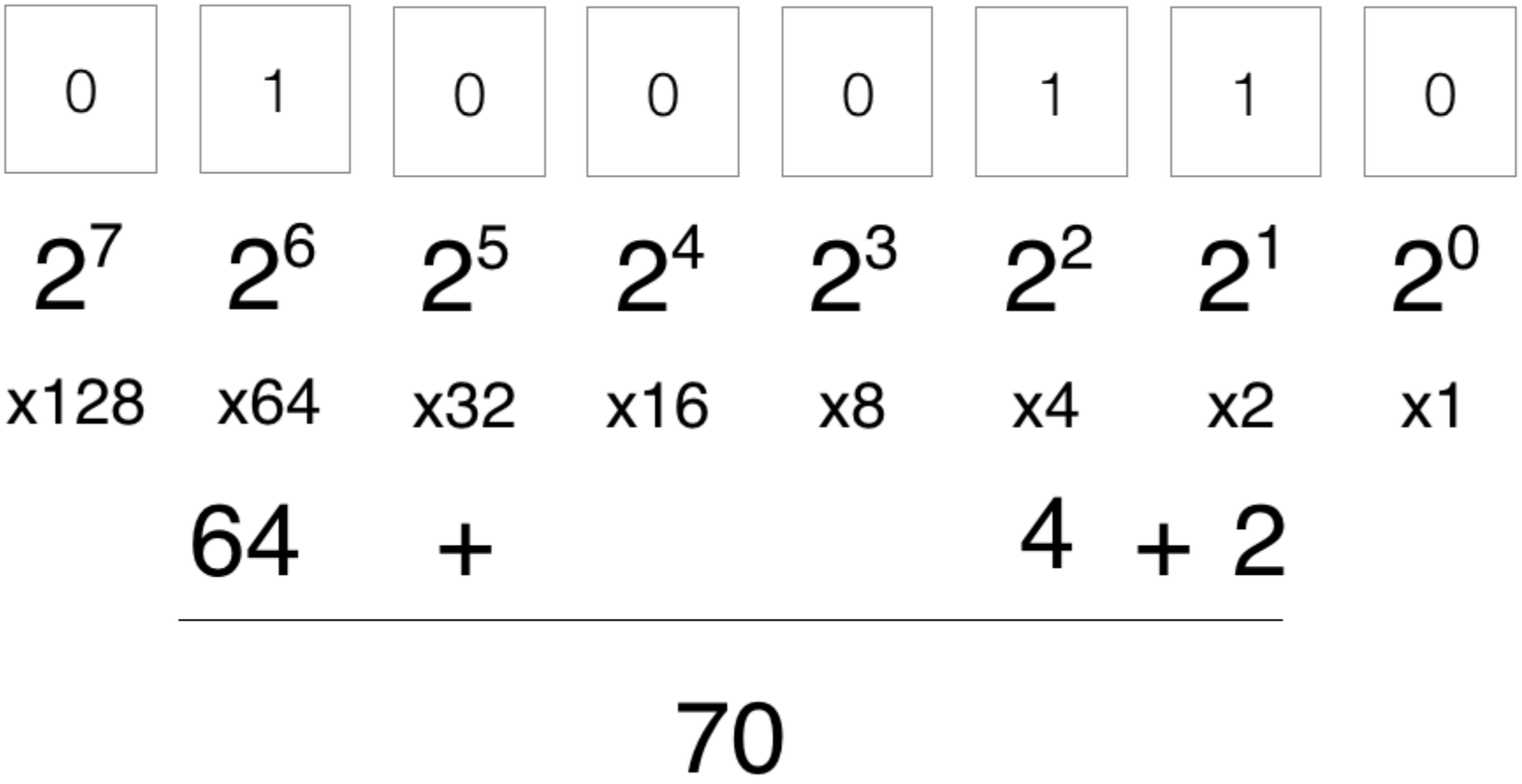
```
bool_1 = not (True or True)
bool_2 = True and (False or False)
bool_3 = True or (False or False)
bool_4 = (True or not True) and (True and True)
bool_5 = (3>5) or (5<4 and not 5>=7)
print(bool_1)
print(bool_2)
print(bool_3)
print(bool_4)
print(bool_5)
```

```
False
False
True
True
False
```

Binary Representation of Decimal Numbers

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Binary Representation of Decimal Numbers



Bitwise operators

Bitwise operators act on operands as if they were string of binary digits.

It operates bit by bit.

- 2 is 10 in binary and 7 is 111.

In the table below:

- Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Bitwise operators

Operator	Description
& Binary AND	Operator copies a bit to the result if it exists in both operands
| Binary OR	It copies a bit if it exists in either operand.
^ Binary XOR	It copies the bit if it is set in one operand but not both.
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.

Bitwise operators

A	B	A | B	A & B	A^ B	~A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Bitwise operators AND OR

5 & 7

- Same as 101 & 111.
- This results in 101
- Which is binary for 5.

4 | 8

- Binary for 4 is 0100, and that for 8 is 1000.
- After operation, output is 1100
- Which is binary for 12.

Bitwise operators AND OR XOR

```
"{0:b}".format(5)
```

```
'101'
```

```
print(5&7)
```

```
5
```

```
print(4|8)
```

```
12
```

```
print(5^3)
```

```
6
```

Shift operator

Left Shift (<<)

```
In [60]: 12<<1
Out[60]: 24

In [61]: 12>>1
Out[61]: 6

In [62]: 12<<2
Out[62]: 48

In [63]: 12>>2
Out[63]: 3
```

Right Shift (>>)

Operator Precedence

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, in, not in	Comparisions, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

DATA TYPE CONVERSION
Explicit and Implicit