1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

10 Type bool and Boolean Expressions

In this section we introduce the Python type `bool` and the operators that create and combine Boolean values.

10.1 Type bool

Python's type for Boolean values in `bool`.

It has two values, written with the keywords `False` and `True`.

```
>>> True
True
>>> False
False
>>> type(True)
<class 'bool'>
>>>
```

10.1.1 The bool constructor

Like any other Python type `bool` is a class, and has a constructor of the same name.

It can be used to convert values of other types to `bool`.

```
>>> bool(0)
False
>>> bool(42)
True
>>> bool("")
False
>>> bool("False")
True
>>>
```

Zero converts to `False`.

Any non-zero number converts to `True`.

Empty strings convert to `False`.

Any non-empty string converts to `True`.

Any value can be converted to a `bool`, explicitly using the `bool` constructor or implicitly by coercion.

10.2 Relational operators

The most common way to create Boolean values is by comparing values with the relational operators.

operator	*meaning*	*expression → result*
`==`	equal to	`1 == 1` → `True` `1 == 2` → `False`
`!=`	not equal to	`1 != 2` → `True`
`<`	less than	`2 < 3` → `True`
`>`	greater than	`3 > 2` → `True`
`<=`	less than or equal to	`2 <= 2` → `True` `2 <= 3` → `True`
`>=`	greater than or equal to	`2 >= 2` → `True` `3 >= 2` → `True`

10.3 Boolean operators

Python's Boolean operators are keywords.

operator	*meaning*	*expression → result*

`and`	and (∧)	`True and True` → `True`
		`True and False` → `False`
		`False and False` → `False`

`or`	inclusive or (∨)	`True or True` → `True`
		`True or False` → `True`
		`False or False` → `False`

`not`	not (¬)	`not True` → `False` `not False` → `True`

10.3.1 short-circuit operators

`and` and `or` don't always need to fully evaluate both of their operands.

Consider the expression p `and` q.

If p is `True`, then the result will depend on q.

But if p is `False`, the result will be `False` regardless of q.

In that case, Python does not bother to evaluate q.

Consider the expression p `or` q.

If p is `False`, then the result will depend on q.

But if p is `True`, the result will be `True` regardless of q.

In that case, Python does not bother to evaluate q.

Short-circuit operators can avoid the evaluation of risky expressions.

10.3.2 Summary of operators and their precedences

highest precedence	`**`
	unary +, unary -
	*, /, //, %
	binary +, binary -
	==, !=, <, >, <=, >=
	`not`
	`and`
lowest precedence	`or`

10.3.3 Example: leap years

We can now write some quite complex Boolean expressions.

> *Problem*: Is a given year a leap year?
>
> A year is a leap year if it is divisible by 4, unless it is divisible by 100, unless it is divisible by 400.

Rewriting that, There are two ways a year can be a leap year:

- The year can be a leap year if:
  - it is divisible by 4; *and*
  - it is not divisible by 100; *or*
- the year is a leap year if it is divisible by 400.

We can tell if one number is divisible by another using modulo.

```
>>> 10 % 3
1
>>> 10 % 5
0
>>> 10 % 3 == 0
False
>>> 10 % 5 == 0
True
>>>
```

Rewriting again, There are two ways a year can be a leap year:

- The year y can be a leap year if:
  - `y % 4 == 0`; *and*
  - `y % 100 != 0`; *or*
- `y % 400 == 0`.

In a program:

```
# file: leap1.py
# Is a year a leap year?

y = int(input("What year? "))
isLeap = (y % 4 == 0 and y % 100 != 0) or y % 400 == 0
print("Is it a leap year?", isLeap)
```

```
$ python3 leap1.py
What year? 2016
Is it a leap year? True
$ python3 leap1.py
What year? 2018
Is it a leap year? False
$ python3 leap1.py
What year? 2000
Is it a leap year? True
$
```

Discuss whether the parentheses in this statement are really needed.

```
isLeap = (y % 4 == 0 and y % 100 != 0) or y % 400 == 0
```

10.4 Python idiosyncrasies

So far, everything in this section is pretty standard for programming languages.

However Python has some unusual semantics when it comes to Booleans and Boolean operators.

10.4.1 Range testing

Unusually for programming languages, in Python a value may be tested to see if it lies within a range, using two relational operators.

```
>>> x = 5.5
>>> 0.0 <= x < 10.0
True
>>>
```

10.4.2 Truth value testing

Almost any value may be used as if it is a Boolean value, according to the truth value testing rules:

For the types we know so far:

Numbers Any zero value can be used as if it is `False`.

Any non-zero value can be used as if it is `True`.

Strings An empty string can be used as if it is `False`.

Any non-empty string can be used as if it is `True`.

10.4.3 is/is not

All values are objects.

== and != test whether two values are equal values or not. The operators `is` and `is not` test whether two values are stored in the same object or not.

```
>>> s = "apple"
>>> t = input()
apple
>>> s == t
True
>>> s is t
False
>>> s is not t
True
>>>
```

10.4.4 and/or always return one of their arguments

Most languages would always return a Boolean value from `and` and `or`.

But Python's `and` and `or` always return one of their arguments.

This can yield some weird results.

```
>>> True or 0
True
>>>
```

The answer is `True` because `or` is a short-circuit operator.

If the first operand is `True` the result is `True`, ignoring the second operand.

```
>>> False or 0
0
>>>
```

Since the first operand is `False`, the result depends on the second operand.

Since Python always returns one of the operands, the result is 0.

```
>>> 0 or 42
42
>>>
```

`0` has the truth value `False`, so the the result is the second operand.

What will be the results of the following expressions?

- `False and "Banana"`
- `"Apple" and True`
- `"Apple" and False`
- `"Apple" and "Banana"`

Test in the REPL.

Discuss: Is this behavior a good thing:

- for professional programmers; and/or
- learning programmers?

10.4.5 Summary of operators and their precedences

highest precedence	`**`
	unary +, unary -
	*, /, //, %
	binary +, binary -
	==, !=, <, >, <=, >=, `is`, `is not`
	`not`
	`and`
lowest precedence	`or`

Section summary

This section covered:

- the type `bool` and its values `True` and `False`; and
- its constructor;
- Python's relational and Boolean operators.