COMP10001 Foundations of Computing
Tuples and Conditionals

Semester 2, 2018
Chris Leckie & Nic Geard

THE UNIVERSITY OF MELBOURNE

Lecture Agenda

- Last lecture:
  - Strings: indexing, slicing and formatting
  - Functions: print, t
  - Comments
- This lecture:
  - Character encodings
  - Tuples
  - Truth conditions

Announcements

- Worksheet 0 is optional...
- Worksheets 1 and 2 due end of this coming Monday (6 August)... don't need to do anything to "submit" your work; just get as many green diamonds as possible by then
- Mid-semester test: 1pm Wednesday 9 September; ie, during lecture timeslot (more details on test, location, etc closer to the date)

Strings and Formatting I

- Often we want to insert variables into strings, optionally with some constraint on how they are formatted/presented
- We can do this in part through string concatenation (+), but it has its limitations:

```
>>> response = "yes"
>>> sentiment = 1/1
>>> print(response + ", " + response + ", " + \
... response + " ... I " + \
... str(100*sentiment) + "% agree")
yes, yes, yes ... I 100.0% agree
```

Strings and Formatting II

- A cleaner, more powerful way is with **format strings** ("f-strings"), marked with an "f" prefix at the start of the string:

```
>>> response = "yes"
>>> sentiment = 1/1
>>> print(f"{response}, {response}, {response}" + \
... " ... I {100 * sentiment:.0f}% agree")
yes, yes, yes ... I 100% agree
```

  - insert variables into strings with braces, possibly with some associated operators (e.g. `100 *`)
  - optionally add formatting specifiers with a colon (":"), e.g. to stipulate the number of decimal places to use for a float (e.g. ".0f" = zero decimal places)

Character Representation

- Computers like bits, and so represent characters as (positive) integer codes
- Python3 defaults to UTF-8 encoding: Unicode, with 8 bits for ASCII, where the character `'A'` has a numerical value of 65, `'B'` is 66, ...
- Code↔character conversion:
  - `ord()`: convert an ASCII character into its code
  - `chr()`: convert an `int` code (0–255) into its corresponding ASCII character
- This is important when we sort strings/check for string "precedence"

Tuples I

- Tuples (`tuple`) are very similar to strings, in that they can be of arbitrary length and can be indexed and sliced...
- However, they can contain more than characters.

```
>>> t = (1.2, 'twine', 3)
>>> t[1]
['twine']
>>> t[0:2]
(1.2, 'twine')
```

Tuples II

- The main places where we will use tuples are:
  - as keys to dictionaries (see later ...)
  - as a way of passing/returning multiple values to functions (see later...)
  - in assignments, e.g.:

```
>>> a = 1; b = 2
>>> print(a,b)
(1, 2)
>>> (a,b) = (b,a)
>>> print(a,b)
(2, 1)
```

What do we know so far?

Syntax

- Maths...
- `print()`, `len()`, `abs()`
- `int`, `float`, `complex`, `str`, tuples
- $*$, + for strings
- Variables, assignment =

Semantics

- Maths expressions are resolved with BODMAS
- Types are important: overloading
- Assignment changes state

In Search of the Truth ...

- Often, we want to check whether a particular value satisfies some condition:
  - does it have four legs?
  - is it over 18?
  - should we add another ’na’ to Hey Jude?
- For this, we require:
  - a way of describing whether the test is satisfied or not
  - a series of comparison operators
  - a series of logic operators for combining comparisons
  - a way of conditioning behaviour on the result of a given test

Capturing Truth: The `bool` Type

- We capture truth via the `bool` (short for "Boolean") type, which takes the two values:
  - `True`
  - `False`
- As with other types, we can "cast" to a `bool` via the `bool()` function:

```
>>> bool(3)
True
>>> bool(0)
False
>>> bool("banana")
True
```

  Every type has a unique value for which `bool()` evaluates to `False` (what are they?)

Evaluating Truth: Comparison

- We evaluate truth via the following Boolean comparison operators:

`==`	equality; NOT the same as =
`>`, `>=`	greater than (or equal to)
`<`, `<=`	less than (or equal to)
`!=`	not equal to
`in`	is an element of

```
>>> 2 == 3
False
>>> 'a' <= 'apple'
True
>>> 2 != 3
True
>>> '3' in '11235'
True
```

Combining Truth

- We combine comparison operators with the following logic operators:
  - `and`, `or`, `not`:

`and`	`True`	`False`
`True`	`True`	`False`
`False`	`False`	`False`

`or`	`True`	`False`
`True`	`True`	`True`
`False`	`True`	`False`

`not`	`True`	`False`
	`False`	`True`

- NB: precedence: `not` > `and` > `or`

Combining Truth: Examples I

```
>>> age = 20
>>> age >= 18
True
>>> tall = True; ears = "rabbit"; back = "grey"
>>> whiskers = True; stomach = "cream"
>>> tall and ears == "rabbit" and back == "grey" \
... and whiskers and stomach == "cream"
True
>>> not False or True
True
>>> not (False or True)
False
>>> year = 2015
>>> 2001 < year and year < 2100
True
>>> 2001 < year < 2100
True
```

Combining Truth: Examples II

- The way logic operators are interpreted in Python is by evaluating the truth value of each operand, and combining them, e.g.:

```
>>> tall and ears == "rabbit" and 3
```

is equivalent to:

```
>>> bool(tall) and bool(ears == "rabbit") \
... and bool(3)
```

Conditioning and Code Blocks

- We can condition the execution of a “block” of code with `if` statements

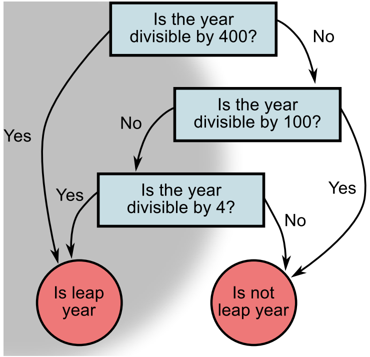
  a “block of code” is a contiguous series of lines of code which are “indented” at (at least) a certain level

```
if␣balance␣-␣withdraw␣>=␣0:
␣␣␣␣balance␣=␣balance␣-␣withdraw
␣␣␣␣print("Withdrawn")
␣␣␣␣if␣balance␣<␣low:
␣␣␣␣␣␣␣␣print("Time␣to␣ring␣mum!")
```

The block only executes if the condition in the `if` statement evaluates to `True`

Conditional Recap

- Problem: evaluate whether a given year is a leap year (`True`) or not (`False`)
- Flowchart:


Is the year divisible by 400?
No
Yes
Is the year divisible by 100?
No
Yes
Is the year divisible by 4?
Yes
No
Is leap year
Is not leap year

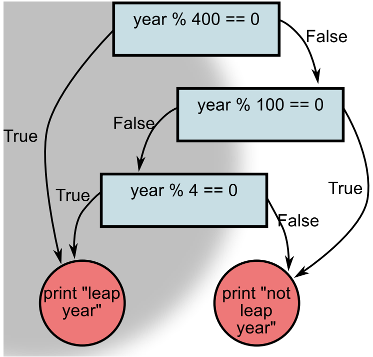
Cascading Conditions

- It is possible to test various **mutually-exclusive** conditions by adding extra conditions with `elif`, and possibly a catch-all final state with `else`

```
if year % 400 == 0:
    print("leap year")
elif year % 100 == 0:
    print("not leap year")
elif year % 4 == 0:
    print("leap year")
else:
    print("not leap year")
```

Conditional Recap

- Problem: evaluate whether a given year is a leap year (`True`) or not (`False`)
- Pythonic flowchart:



Class Exercise

- Simply the preceding code into one `if` statement and one `else` statement (and no `elif` statements)

Lecture Summary

- What is a `tuple`, and how does it relate to a list?
- What is the `bool` type?
- What Boolean comparison operators are commonly used in Python?
- What logic operators are commonly used in Python? What is the operator precedence?
- What are `if` statements and code blocks?
- How can you cascade conditions in Python?