THE UNIVERSITY OF MELBOURNE

COMP10001

Foundations of Computing

Semester 1, 2021

Tutorial 3

Andrew Naughton

andrew.naughton@unimelb.edu.au

THE UNIVERSITY OF MELBOURNE

Outline

- Booleans
- Operators & Precedence
- if Statements
- Sequences: Indexing & Slicing
- Functions
- Exercises
- Problems (time permitting)

THE UNIVERSITY OF MELBOURNE

Booleans

- Can other types be converted to bools?
  - int:
    - 0 converts to False
    - All other ints convert to True
  - float:
    - 0.0 converts to False
    - All other floats convert to True
  - str:
    - “” (empty string) converts to False
    - All other strs convert to True

THE UNIVERSITY OF MELBOURNE

Relational Operators

- Compares two values and produces a Boolean result (True or False)

Operator	Meaning
==	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

THE UNIVERSITY OF MELBOURNE

Logical Operators

- Combine Boolean values to return a single truth value
- And
  - Requires both operands to be True to return True; False otherwise
  - E.g.
    - True and True -> True
    - True and False -> False

THE UNIVERSITY OF MELBOURNE

Logical Operators

- Or
  - Requires at least one operand to be True to return True; False otherwise
  - E.g.
    - True or True -> True
    - True or False -> True
- Not
  - Inverts a truth value
  - E.g.
    - not True -> False
    - not False -> True

THE UNIVERSITY OF MELBOURNE

Order of Precedence

- In order of decreasing priority:
  - Relational operators
  - Not
  - And
  - Or
- Brackets can be used to clarify the order of operations as well

THE UNIVERSITY OF MELBOURNE

if Statements

- Skeleton

```
if <condition> :
    # do something
elif <condition> :
    # do something else
else:
    # do something else
```

THE UNIVERSITY OF MELBOURNE

Sequences

- Those data types that allow us to store a series of objects in a particular order
- str
  - Stores a sequence of characters
- list, tuple
  - Stores a sequence of any type of object (e.g. list of ints)

THE UNIVERSITY OF MELBOURNE

Sequences: Indexing

- To access the element stored in a particular (integer) position in a sequence, aka “index”
- Trying to access an element at an index that doesn’t exist produces an IndexError

I	*t*		*w*	*a*	*s*		*a*		*d*	*a*	*r*	*k*
0	1	2	3	4	5	6	7	8	9	10	11	12
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

THE UNIVERSITY OF MELBOURNE

Sequences: Slicing

- Allows us to slice/extract a subsection of a sequence
- Has the form:
  - <var or literal>[<start>:<stop>:<step size>]
  - Where
    - *start*: index to start slicing at (included)
    - *stop*: index to stop slicing at (excluded)
    - *Step size*: number of elements to move over by when slicing

THE UNIVERSITY OF MELBOURNE

Sequences: Slicing

- Note
  - Slicing always returns a sequence (IndexError N/A)
  - If *start* is not explicitly stated, default is 0*
  - If *end* is not explicitly stated, default is sequence length*
  - If *step size* is not explicitly stated, default is 1

* Assuming *step size* is positive

THE UNIVERSITY OF MELBOURNE

Functions

- Skeleton

```
def function_name(arg1, arg2):
    # do something
    return something
```

- Using return is optional; returns None by default
- Brackets are always needed to call a function
- Without them, we are merely holding a reference to the function

Functions

- How do they help?
  - Reduce code duplication
  - Makes our code easier to edit and maintain
  - Can be used elsewhere and in more places
    - Modular code: flexible – i.e. not hardcoded – and variety of use
- Without functions, code would be extremely messy and error-prone

THE UNIVERSITY OF
MELBOURNE

Exercises