COMP10001 Foundations of Computing
Functions, Methods, Comments, and Tuples

Semester 1, 2019
Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

THE UNIVERSITY OF MELBOURNE

— version: 1486, date: March 21, 2019 —

Lecture Agenda

- Last lecture — Grok Worksheets 3, 5
  - Conditionals (cont.)
  - Functions
- This lecture — Grok Worksheet 5
  - Functions (cont.)
  - Methods
  - Comments
  - Tuples

Announcements

- Worksheets 3 and 4 due end of this coming Monday
- First guest lecture tomorrow

Lecture Outline

1. Functions (cont.)
2. Methods
3. Comments
4. Tuples

Functions: More Details

- It is possible to define "variable-arity" functions (i.e. functions which take variable numbers of arguments) by specifying default values for arguments:

```
def seconds_in_year(days=365):
    return days * 24 * 60 * 60
```

```
>>> seconds_in_year()
31536000
>>> seconds_in_year(366)
31622400
```

Variables and "Scope"

- Each function (call) defines its own local variable "scope". Its variables are not accessible from outside the function (call)

```
def subtract_one(k):
    k = k - 1
    return k

i = 0
n = subtract_one(i)
print(i)
print(n)
print(k)
```

Variables and "Scope"

- Are the semantics different to the previous slide?

```
def subtract_one(i):
    i = i - 1
    return i

i = 0
n = subtract_one(i)
print(i)
print(n)
print(k)
```

Variables and "Scope"

- Functions can access variables defined outside functions ("global" variables), although they should be used with extreme caution (perhaps never!)

```
def fun1(j):
    fun2(j)
    return 1
def fun2(k):
    global i, j  # global variables
    i = j = k = k + 1
    return 2
i = j = k = 1
fun1(i)
```

Reasons for Using Functions

- "Archiving" code in libraries
- Removing redundancy
- Ease of testing
- Increasing modularity
- Increasing readability

Lecture Outline

1 Functions (cont.)

2 Methods

3 Comments

4 Tuples

Functions and Methods

- Functions and methods provide pre-defined functionality over a pre-defined set of arguments (generally of fixed type), in the form of a predefined set of outputs
- **Functions** share the same namespace as variables, and are called as "standalones"

```
>>> type(len)
<type 'builtin_function_or_method'>
>>> len("a piece of string")
17
```

Functions and Methods

- **Methods** are defined for/called from objects of a given type, and are called as `object.METHOD()` from objects of that type

```
>>> type(upper)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'upper' is not defined
>>> "a piece of string".upper()
'A PIECE OF STRING'
```

- *Function or method, that is the question*: the question of whether to define a procedure as a function or method can be very subtle (cf. `len()`)

Other Useful Methods for Strings

- Return s in all uppercase:

```
s.upper()
```

- Return s in all lowercase:

```
s.lower()
```

- Return s with all instances of characters in STRING (whitespace if STRING is not supplied) removed from start and end of s

```
s.strip(STRING)
```

Lecture Outline

1. Functions (cont.)
2. Methods
3. Comments
4. Tuples

Comments

- Comments are notes of explanation that document lines or sections of a program, which follow a # (hash) character
- Python ignores anything following a # on a single line (multi-line "commenting" possible with """):

```
# OK, here goes
"""Three blind mice,
Three blind mice,
..."""
print("Hello world")
```

Commenting Expectations

- For this subject we require:
  - All key variables should have comments about what they are used for (as should user-defined functions)
  - Your code should describe **why** you do things, not **what** you do
  - Commenting can also be used to stop lines of code from being executed. This is called "commenting out" code.

Lecture Outline

1. Functions (cont.)
2. Methods
3. Comments
4. Tuples

Keeping it together: Tuples

- Tuples are just like strings but:
  - each element can be something other than a character
  - we use (,) rather than " " to build them

```
>>> costs = (1, 2.6, 7.1, -3.14)
>>> print(costs[0])
1
>>> print(costs[2:4])
(7.1, -3.14)
```

When would I Use Tuples?

- Representing "multi-variate" objects:
  - representing coordinates `(x, y, z)`
  - health records `(name, address, ...)`
  - playing cards `(value, suit)`
  - map positions `(latitude, longitude)`
  - mental state `(love, hate, desire, beliefs, ...)`
  - limb positions `(angle, voltage, resistance)`

Useful Coding Applications of Tuples

- To return multiple values:
  `return (name, age, gender)`
- To swap values between variables:
  `(a, b) = (b, a)`
- To test for one of a series of values:
  `number in (12, 1, 2)`
- As keys to dictionaries (see later ...)

Just like Strings, Tuples are "Immutable"

- Once they are created, you cannot change elements

```
>>> data = (1, True, 'alice', 'bob')
>>> data[0] = 0
TypeError: 'tuple' object does not support ...
>>> data = "Alice and Bob"
>>> data[0] = 'H'
TypeError: 'str' object does not support ...
```

Variable-arity Functions: Redux

- A second way of defining a "variable-arity" function is by identifying a parameter as generating a variable-sized tuple of any "leftover" arguments:

```
def varfun(num, *rest):
    return (num, rest)
```

```
>>> varfun(1, 2)
(1, (2,))
>>> varfun(1)
(1, ())
>>> varfun(1, 2, 3)
(1, (2, 3))
```

Lecture Summary

- How do we define variable-arity functions in Python?
- What are the reasons we define functions?
- What are methods, and how are they similar/different to functions?
- Comments: what and how?
- What is a tuple?