COMP10001 Foundations of Computing
Variables and Strings

Semester 2, 2021

Chris Leckie, Marion Zalk and Farah Khan

THE UNIVERSITY OF MELBOURNE

— VERSION: 1478, DATE: MARCH 15, 2019 —

Lecture Agenda

- Last lecture — Grok Worksheet 1
  - Literals
  - Types
- This lecture — Grok Worksheets 1, 6
  - Variables and assignment
  - String basics

Lecture Outline

Literals and Variables

- To date, all of the values have taken the form of "literals", i.e. the value is fixed and has invariant semantics
- It is also possible to store values in "variables" of arbitrary name via "assignment" (=)
  - N.B. "=" is the assignment operator and NOT used to test mathematical equality (we'll get to that later ...)
- We use variables to name cells in the computers memory so we don't need to know their addresses

The Ins and Outs of Assignment

- The way assignment works is the right-hand side is first "evaluated", and the value is then assigned to the left-hand side ... making it possible to assign a value to a variable using the original value of that same variable:

```
>>> a = 1
>>> print(a + 1)
2
>>> print(a + a + 1)
3
>>> a = a + a + 1
>>> print(a)
3
```

The Ins and Outs of Assignment

- Note that assignment can only be to a single object (on the left-hand side):

```
>>> a = 1
>>> a = a + a + 1
>>> a + 1 = 2
  File "<stdin>", line 1
SyntaxError: can't assign to operator
```

... although we will later see that it is possible for an object to have complex structure, and that it is possible to assign to the "parts" of an object ...

The Ins and Outs of Assignment

- It is also possible to assign the same evaluated result to multiple variables by "stacking" assignment variables:

```
>>> a = b = c = 1
>>> a = b = c = a + b + c
>>> print(a)
3
>>> print(b)
3
>>> print(c)
3
```

Class Exercise

- Python is an "imperative" language, meaning that it has "program state" and the values of variables are changed only through (re-)assignment:

```
>>> a = 1
>>> b = 2
>>> a = a + 1
>>> b = b + a
>>> print(a)
>>> print(b)
```

What is the output of this code?

Variable Naming Conventions

- Variable names must start with a character (`a-zA-Z`) or underscore (`_`), and consist of only alphanumeric (`0-9a-zA-Z`) characters and underscores (`_`)
- Casing is significant (i.e. `apple` and `Apple` are different variables)
- "Reserved words" (operators, literals and built-in functions) cannot be used for variable names (e.g. `in`, `print`, `not`, ...)
  - valid variable names: `a`, `dude123`, `_CamelCasing`
  - invalid variable names: `1`, `a-z`, `13CABS`, `in`

Class Exercise

- Calculate the *i*th Fibonacci number using only three variables

```
fn_2 = 0
fn_1 = 1
fn = fn_1 + fn_2
print(fn)
fn_2 = fn_1
fn_1 = fn
fn = fn_1 + fn_2
print(fn)
```

Lecture Outline

A New Type: Strings

- A string (`str`) is a "chunk" of text, standardly enclosed within either single or double quotes:
  - `"Hello world"`
  - `'How much wood could a woodchuck chuck'`
- To include quotation marks (and slashes) in a string, "escape" them (prefix them with `\`):
  - `\"`, `\'` and `\\`
- Also special characters for formatting:
  - `\t` (tab), `\n` (newline)
- Use triple quotes (`'` or `"`) to avoid escaping/special characters:
  - `"""Ow," he said/yelled."""`

String Operators

- The main binary operators which can be applied to strings are:
  - + (concatenation)

```
>>> print("a" + "b")
ab
```

  - * (repeat string *N* times)

```
>>> print('z' * 20)
zzzzzzzzzzzzzzzzzzzz
```

  - `in` (subset ... see next lecture for details)

```
>>> print('z' in 'zizzer zazzer zuzz')
True
```

Overloading

- But but but ... didn't + and * mean different things for `int` and `float`?
  - Answer: yes; the operator is "overloaded" and functions differently depending on the type of the operands:

```
>>> print(1 + 1)
2
>>> print(1 + 1.0)
2.0
>>> print("a" + "b")
ab
>>> print(1 + 'a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Functions Applicable to Strings

- Useful functions related to strings:
  - `len` (calculate the length of the string)

```
>>> print(len("a piece of string"))
17
```

  - `str` (convert an object to a string)

```
>>> str(2)
'2'
>>> str(2.0)
'2.0'
>>> str("string")
'string'
```

Class Exercise

- Given `num` containing an `int`, calculate the number of digits in it

```
>>> num = 3000
>>> len(str(num))4
>>> str(num)'3000'
>>> num = -20
>>> len(str(num))3
>>> str(num)'-20'
>>> abs(num)20
>>> len(str(abs(num)))2
>>> print(len(str(abs(num))))2
```

Strings and Formatting

- 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

- 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)

Lecture Summary

- Assignment: what is it, and how does it work?
- Strings: how are they formatted, and what operations/functions can be applied to them?