COMP10001 Foundations of Computing
Strings and Variables

Semester 2, 2018
Chris Leckie & Nic Geard

Lecture Agenda

- Last lecture:
    - Grok
    - Types
- This lecture:
    - Strings
    - Literals, variables and assignment
    - Type conversion
    - Printing
    - Comments

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)

    ```
    >>> 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 "<web session>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

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 I

- 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 valuable 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 II

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

```
>>> a=1
>>> a=a+a+1
>>> a+1=2
Traceback (most recent call last):
  File "<web session>", 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 III

- 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
```

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 (1)

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

Assignment and State

- 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
>>> c = a + b
>>> a = 2
>>> print(c)
3
>>> c = a + b
>>> print(c)
4
```

Type Conversion

- Python implicitly determines the type of each literal and variable, based on its syntax (literals) or the type of the assigned value (variables)
- To "cast" a literal/variable to a different type, we use functions of the same name as the type: `int()`, `float()`, `str()`, `complex()`

```
>>> print(float(1))
1.0
>>> print(int(1.0))
1
>>> print(int(1.5))
1
>>> int('a')
Traceback (most recent call last):
  File "<web session>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
```

A Couple of Other Useful Functions

- `abs()`: return the absolute value of the operand
- `len()`: return the length of the iterable operand (i.e. a `str` for now)

```
>>> print(len('apple'))
5
>>> print(len(1))
Traceback (most recent call last):
  File "<web session>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

Class Exercise (2)

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

The print() Function

- The `print()` function can be used to print the value of the operand (of any type)

```
>>> a = 1
>>> print(a)
1
```

- In the console, there is no noticeable difference between printing and executing a variable:

```
>>> a = 1
>>> print(a)
1
>>> a
1
```

but when you “run” code from a file, you will only see the output of `print()` functions

The print Statement

- In Python 2, you can use either the `print` statement (`print ...`) or the `print` function `print(...)`, but Python 3 only allows the `print` function

```
>>> a = 1
>>> print(a)
1
>>> print a    # Python 2
1
```

so if you use Python 2 code from the www, remember to convert print statements to print functions.

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 `"""`):

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

Commenting Expectations

- For this subject we require:
  - A set of comments at the beginning of every python program:

    ```
    # What does this program do
    # Author(s): Who wrote me
    # Date created
    # Date modified and reason
    ```

  - All key variables should have comments about what they are used for (as should user-defined functions)
  - Commenting can also be used to stop lines of code from being executed. This is called “commenting out” code.

More on String Manipulation

- As well as “assembling” strings via + and ∗, we are able to pull strings apart in the following ways:
    - “indexing” — return the single character at a particular location
    - “slicing” — extract a substring of arbitrary length
    - “splitting” — break up a string into components based on particular substrings

String Manipulation: Indexing

- Each character in a string can be accessed via "indexing" relative to its position from the left of the string (zero-offset) or the right of the string ([minus] one-offset):

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

```
>>> story[-8]
's'
>>> story[5]
's'
```

String Manipulation: Slicing

- It is possible to “slice” a string by specifying a left (`L`) and (non-inclusive) right (`R`) `int` value:

```
>>> story[1:11]
't was a da'
```

N.B. the sliced substring length = `R` − `L`

- By default, `L=0` and `R` is the length of the string:

```
>>> story[:-7]
'It was'
```

- It is also possible to specify slice “direction”:

```
>>> story[:-7:-1]
'krad a'
```

Class Exercise (2)

- Generate the “middle half” of a given string

Lecture Summary

- Strings: how are they specified, and what basic operators apply to them?
- Literals and variables: what's the difference, and what are the constraints on variable names?
- Type conversion: what and how?
- Strings: what are indexing, slicing and splitting? how do we format strings?
- Comments: what and how?