COMP10001 Foundations of Computing
The Basics of Types

Semester 2, 2016
Chris Leckie

July 8, 2016

Lecture Agenda

- Last lecture:
  - Basics of Python
  - Grok
- This lecture:
  - Design of algorithms
  - Types
  - Strings
  - Literals and variables

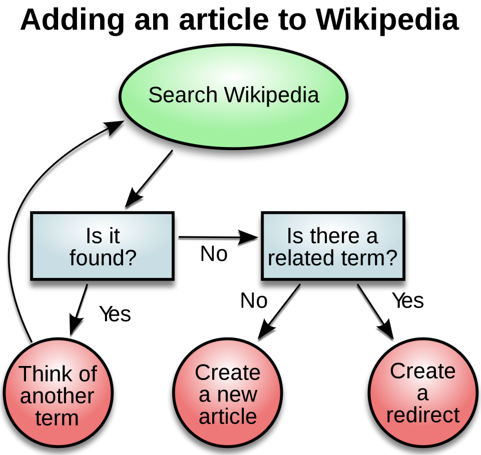
Programming

- Computer programs are simply sets of steps to complete some task
- Determining what the steps should be requires learning how computers "think"...
- ...and how a particular programming language expresses the way a computer thinks.
- Communicating with humans in different languages can be tricky, but achievable. Computers are not so forgiving.

Program Design

- Many modern computing languages express similar concepts
- They allow "conditioning" on particular values, "looping" over sub-sets of steps, and "nesting" of loops
- Common ways to abstractly represent programs are:
    - flowcharts
    - pseudo-code (i.e. a computer program in an abstract language, without the "bookkeeping" that individual languages require) `http://www.bestrecipes.com.au/recipe/choc-chip-cookies-L4351.html`

Example Flowchart

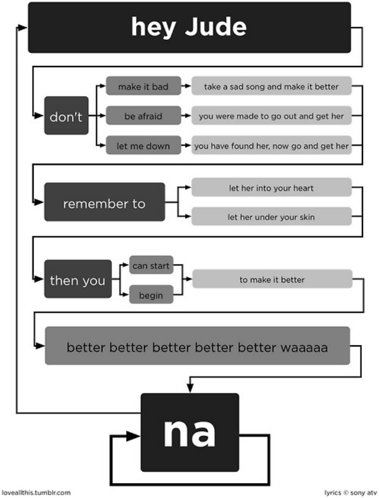


Source(s): `http://commons.wikimedia.org/wiki/File:Wikipedia_article-creation-2.svg`

Equivalent Psuedocode

```
repeat
   search Wikipedia for the candidate article
   if article found then
      think of another term
   else if article found for related term then
      create a redirect
   else
      create a new article
   end if
until article created or redirect created
```

More Interesting Flowchart



Source(s): `http://laughingsquid.com/hey-jude-flow-chart/`

Types (1)

- In Python, every object has a “type”, which defines: (a) what operators, “functions” and “methods” can be applied to it, and (b) the semantics of each; consider:
- The two number types we will see most of are:
  - `int` (integer)
  - `float` (real number)

  also `complex` for complex numbers
- So how does Python work out the type for a given (real) number? If it contains a decimal place (.), it's a `float`, otherwise it's an `int`

Types (2)

- Use the “function” `type()` to determine the type of an object:

```
>>> print(type(1))
<type 'int'>
>>> print(type(1.0))
<type 'float'>
```

- The semantics of operators and functions is determined by the types of the operands:

```
>>> print(type(1 + 2))
<type 'int'>
>>> print(type(1/2))
<type 'float'>
```

Jargon alert

Syntax: ”the arrangement of words and phrases to create well-formed sentences in a language”

- `print hello''()` Incorrect syntax
- `print('hello')` Correct syntax

Semantics: ”the meaning of a word, phrase, or text”

- `print(1 + 2)` ’+’ adds two numbers
- `print('h'+ 'a')` ’+’ concatenate two strings

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

Looking Towards Next Week

- Commencement of workshops (work out when your workshop is and where your room is located)
- Make sure you can log in to Grok

Lecture Summary

- Types: what are they, what basic types have we learned, and how do you determine the type of a literal/variable?
- 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?