CSCA08

Week 2 - Expressions, Variables and Functions

Brian Harrington & Marzieh Ahmadzadeh

University of Toronto Scarborough

September 11-15, 2017

Admin Stuff

- Exercise 0 is now out
  - Due Friday
- Logging into MarkUs
- Discussion Board
- Installing Software
- Practicals
- Break Bonus
- Facilitated Study Groups

Expressions

- Default line of Python code
- Python tries to resolve every expression into a value
- Expressions can have sub-expressions, which can in turn have sub-sub expressions, etc.

Types

- There are a few basic **types** that a value can have in Python:
  - Numbers (int or float)
  - String (just a special way of saying a line of text)
  - Boolean (True or False)
  - NoneType (A special type for denoting we don't want anything here)
- Unlike other programming languages, Python decides the types as you go
  - This has its benefits and drawbacks
  - We can force Python to change the type of something by **casting** it
- We can find out the type of an expression using the `type()` function

VARIABLES

- So far, each time we've evaluated something, we've either printed it, or just thrown it away
- What if we want to take the result of an expression, and use it over again?
- We can store the result of an expression in a **variable**

Assignment Statements

- Form: `variable` = `expression`
- `expression` is any valid expression, just like what we've been creating
- Instead of printing or returning the result to the shell, we can store it in `variable`
- Next time we use that variable, it will have the stored value in it

Functions

- What if we want to perform the same operation over and over again?
- Not very efficient to cut & paste
  - What if we decide we want to change afterwards?
- We can create a `function`

Functions

- A `function` is just a block of code that we can call when we need it
- Functions take variables as input, and and performs operations based on the values of those variables
- Much like a Mathematical function
- As far as Python is concerned, a function is an expression (i.e., it must be resolved into a single value)

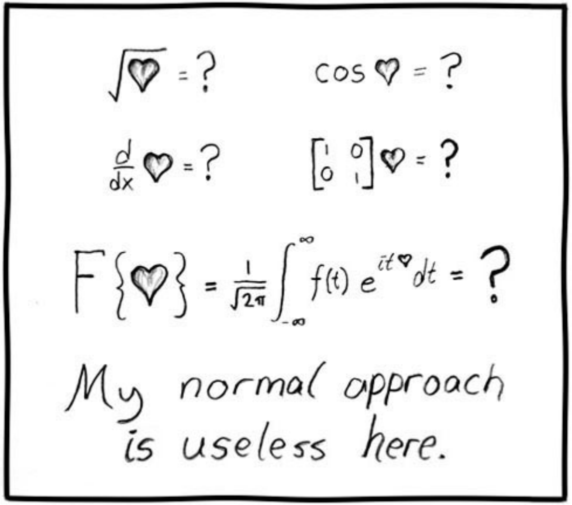
FUNCTIONS

- To define a function
  - `def function_name(parameters):`
    ```
        block
    ```
  - `def` is a python keyword (can’t name a variable def)
  - `parameters` is a list of 0 or more parameters
  - `block` is a block of code
    - MUST be indented
  - Can use `return` to return a value
    - Makes the entire function resolve to that return value

Back to our Example

- We can make it better!

Break


√♡ = ?
cos ♡ = ?
d/dx ♡ = ?
[1 0; 0 1] ♡ = ?
F{♡} = 1/√2π ∫_{-∞}^{∞} f(t) e^{it♡} dt = ?
My normal approach is useless here.

Scope

- Global Variables
    - Created outside function
    - Persist between function calls
    - Can’t be changed from inside a function
- Local Variables
    - Created inside a function
    - Exist only as long as the function is being executed
    - Recreated if the function is called again
    - Parameters are local variables
    - Python checks for local variables first

Tracing

- Tracing code is one of the most important skills you'll learn in the first few weeks of A08
  - Usually ~25-50% of midterm
- Memory model:
  - Make it easier to follow what's happening
  - Understand what's actually going on "under the hood"
  - If you understand the memory model, I can't trick you
  - Nothing is complicated (remember, stupid computers are doing it)