Tip 49: return multiple items from a function

Ben Klemens

7 January 2012

level: Your functions are getting complex
purpose: separate inputs from outputs

A mathematical function doesn't have to map to one dimension: mapping to $\Re^2$ is nothing at all spectacular.

Python lets you return lists, like this:

```
#Given the standard paper size name, return its width, height
def length_width(papertype):
    if (papertype=="A4"):
        return [210, 297]
    if (papertype=="Letter"):
        return [216, 279]
    if (papertype=="Legal"):
        return [216, 356]


[a, b] = length_width("A4");
print("width= %i, height=%i" %(a, b))
```

C won't provide nearly as neat a format, but you can always return a struct, and thus as many subelements as desired. Which is why I was praising the joys of having throwaway structs in the last tip: generating a function-specific struct is not a big deal.

Let's face it: C is still going to be more verbose than languages that have the built-in ability to retun lists. But it is not impossible to clearly express that the function is returning a value in $\Re^2$.

I decided to write the code sample using the `condition ? iftrue : else` form, which is a single expression, and so can appear after the `return`. Notice how it cascades neatly into a sequence of else-if cases (including that last catch-all else clause at the end). I like to format this sort of thing into a nice little table; you can find people who call this terrible style.

```
#include <stdio.h>
#include <math.h> //NaN

typedef struct {
    double width, height;
} size;

size width_height(char *papertype){
  return
    !strcasecmp(papertype, "A4")      ? (size) {.width=210, .height=297}
  : !strcasecmp(papertype, "Letter") ? (size) {.width=216, .height=279}
  : !strcasecmp(papertype, "Legal")  ? (size) {.width=216, .height=356}
                                     : (size) {.width=NAN, .height=NAN};
}

int main(){
    size a4size = width_height("a4");
    printf("width= %g, height=%g\n", a4size.width, a4size.height);
}
```

The alternative is to use pointers, which is darn common and not considered bad form, but it certainly obfuscates what is input and what is output, and makes the version with the extra `typedef` look stylistically great:

```
//return height and width via pointer:
void width_height(char *papertype, double *width, double *height);
```

or

```
//return width directly and height via pointer:
double width_height(char *papertype, double *height);
```