1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

8 Programs That Read

So far we have used the Python REPL as a calculator, or used the interpreter to write script that print.

In the REPL we can interactively enter new values, but our scripts always print the same thing, because they can't accept inputs.

Such programs are not very useful.

With this section, we'll fix that.

Along the way we need to learn how to convert data from one type to another.

8.1 Problem: square a number

Let’s start with a very simple problem.

> Write a program that asks for a number then prints the square of the number.

8.1.1 Refine the specification

The problem as given is a bit weakly specified.

If you were providing this program to a paying client or boss, your first question should be, “Exactly what should it look like when it is run?”

Suppose the answer is this:

```
Enter a number: 3
The square of the number = 9.0
```

We can infer:

- what the prompt to the user to enter input should be; and
- that the numbers should be interpreted as `floats`.

Note: in workshops, when we give you problems to solve with Python programs, if you don't think you have been given enough information, your job is to ask for more.

This is what a client or boss would expect you to do.

A *good* programmer writing a *great* program which was *not* actually the program that was required has wasted time and money.

Asking questions is part of the job!

The responsibility is then to follow the requirements as closely as possible.

8.1.2 Decompose the problem

This is how we can *design* a solution to the problem.

Here are the things that should happen in order:

1. Print “`Enter a number:`”.
2. Read the number entered by the user.
3. Print “`The square of the number = `”.
4. Print the square of the number.

We know how to print and calculate stuff, so the only new bit is reading.

8.1.3 Implementation

Implementation is expressing the solution in programming language code.

Start by creating a script file with descriptive comments at the top.

```
# file: square1.py
# Print the square of a number input by the user.
```

We can implement the parts in any order.

Starting with the output at the bottom is very often a good strategy.

Make up a variable that will hold the input number, and print its square with a label.

```
# file: square2.py
# Print the square of a number input by the user.

print("The square of the number =", x * x)
```

Of course if we run it, we'll get an error.

```
$ python3 square2.py
Traceback (most recent call last):
  File "square2.py", line 4, in <module>
    print("The square of the number =", x * x)
NameError: name 'x' is not defined
$
```

If we define the variable with some temporary code, we can test the printing.

```
# file: square3.py
# Print the square of a number input by the user.

x = 3.0
print("The square of the number =", x * x)
```

```
$ python3 square3.py
The square of the number = 9.0
$
```

So how do we read the number?

See the *Built-in Functions* section of the standard library.

The `input` function does what we want. It:

- prints the argument, if it is present, as a prompt to the user; then
- inputs a line of text typed by the user.

Try this:

```
# file: square4.py
# Print the square of a number input by the user.

x = input("Enter a number: ")
print("The square of the number =", x * x)
```

```
$ python3 square4.py
Enter a number: 3.0
Traceback (most recent call last):
  File "square4.py", line 5, in <module>
    print("The square of the number =", x * x)
TypeError: can't multiply sequence by non-int of type
'str'
$
```

The statement that called `input` worked!

But the `print` statement failed, because `x` as type `str`, and two strings can't be multiplied together.

We need to convert the `str` value returned by `input` to a `float`.

It is very important to note that even though we do not explicitly define the types of variables in Python, the key to getting programs to work is to know what type every value and variable is.

Back to the standard library, where the `float` function does what we want.

It will convert a string to a `float`.

We can complete the program by applying the `float` function to the output of the `input` function.

```
# file: square5.py
# Print the square of a number input by the user.

x = float(input("Enter a number: "))
print("The square of the number =", x * x)
```

```
$ python3 square5.py
Enter a number: 3.0
The square of the number = 9.0
$ python3 square5.py
Enter a number: 3
The square of the number = 9.0
$
```

8.2 Constructors as converters

The `float` function can be thought of as a function for converting strings or `int`s to `float`s.

But remember that all values in Python are objects.

So the `float` function must create a new `float` object.

Each type in Python has function of the same name.

That function is the *constructor* (or *initialiser*) for that type.

Constructors are a common feature of object-oriented programming languages.

For each of the types we know so far, `int`, `float`, and `str`, there is a constructor, and it is used to create new objects of its type, converted from its argument.

Examples:

```
>>> float('123.7')
123.7
>>> str(4.2)
'4.2'
>>> float(3)
3.0
>>> int('45')
45
>>> int(4.2)
4
>>> int(4.7)
4
>>>
```

8.3 Problem: molecular weights

Carbohydrates (carbs, *e.g.* glucose, $C_6H_{12}O_6$) are molecules made up of only Hydrogen, Carbon, and Oxygen.

Write a program that can calculate the molecular weight of any carbohydrate.

```
How many atoms of H: 12
How many atoms of C: 6
How many atoms of O: 6
Molecular weight = 180.14586
```

8.3.1 Implementation

Start with a header comment.

```
# file: carbs1.py
# Calculate the molecular weight of any carbohydrate.
```

Consult Professor Google for the atomic weights.

Define constants for each.

A comment explains what the purpose of the symbols we have just made up is.

```
# atomic weights
H = 1.00784
C = 12.0096
O = 15.99903
```

Now read the inputs, with prompts, converting them to `ints`, and print the answer.

```
h = int(input("How many atoms of H: "))
c = int(input("How many atoms of C: "))
o = int(input("How many atoms of O: "))
print("Molecular weight =", h * H + c * C + o * O)
```

```
$ python3 carbs1.py
How many atoms of H: 12
How many atoms of C: 6
How many atoms of O: 6
Molecular weight = 180.15586000000003
$
```

Consult Professor Google again to verify the result.

Section summary

This section covered:

- the `input` function function for prompting for and reading input typed by the user;
- the constructors for `int`, `float`, and `str` to convert values from one type to another; and
- how to use them to create interactive scripts that perform simple calculations.