Python Practice Tasks

J E Paton

June 27, 2016

1 Variables

1. The assignment operator += is used to modify (by incrementing by the value on the right) an already defined variable. Write a program that defines a variable, n, and then increases its value by 1.

Solution:

```
n = 1
n += 1
```

2. Write a program that asks for the user's name and then insults them.

Solution:

```
user_name = raw_input("enter your name: ")
print user_name, "you are a foolish scoundrel."
```

3. Write a program that asks a user for a number and prints the square of that number.

Solution:

```
number = float(raw_input("enter a number: "))
print "x^2=", number**2
```

4. Write a program that prints the roots of a quadratic equation if given, a, b, and c, where

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.1}$$

Solution:

```
a = float(raw_input("enter a: "))
b = float(raw_input("enter b: "))
c = float(raw_input("enter c: "))

print "x=", (-b + (b**2 - 4 * a * c)**0.5) / (2 * a)
print "x=", (-b - (b**2 - 4 * a * c)**0.5) / (2 * a)
```

5. Modify the above program to print the roots of the quadratic equation if given q, e, r, and k where $(qx + e)(rx + k)$.

Solution: Firstly we need to do some maths to express x in terms of the variables given. There are two main ways to do this, using the quadratic formula or by setting brackets to zero. They give the same result.

Method 1: Quadratic Formula

Multiply the brackets.

$$0 = (qx + e)(rx + k) \tag{1.2}$$
$$0 = qrx^2 + qkx + erx + ek \tag{1.3}$$
$$0 = qrx^2 + (qk + er)x + ek \tag{1.4}$$

which if we compare to the quadratic equation, we get

$$a = qr \tag{1.5}$$
$$b = qk + er \tag{1.6}$$
$$c = ek \tag{1.7}$$

which we can put in the quadratic formula,

$$x = \frac{-qk - er \pm \sqrt{(qk + er)^2 - 4qrek}}{2qr} \tag{1.8}$$
$$x = \frac{-qk - er}{2qr} \pm \frac{\sqrt{q^2k^2 + 2qker + e^2r^2 - 4qker}}{2qr} \tag{1.9}$$
$$x = \frac{-qk - er}{2qr} \pm \frac{\sqrt{q^2k^2 - 2qker + e^2r^2}}{2qr} \tag{1.10}$$

which can be simplified using $(m - p)^2 = m^2 - 2mp + p^2$,

$$x = \frac{-qk - er}{2qr} \pm \frac{\sqrt{(qk - er)^2}}{2qr} \tag{1.11}$$
$$x = \frac{-qk - er}{2qr} \pm \frac{qk - er}{2qr} \tag{1.12}$$

which gives the two solutions as

$$x = \frac{-2er}{2qr} \tag{1.13}$$
$$x = \frac{-2qk}{2qr} \tag{1.14}$$

which simplify to

$$x = \frac{-e}{q} \tag{1.15}$$
$$x = \frac{-k}{r} \tag{1.16}$$

Method 2: Setting Brackets to 0

By setting one bracket to zero we can find an expression for x,

$$0 = (qx + e)(rx + k) \tag{1.17}$$
$$0 = qx + e \tag{1.18}$$
$$-e = qx \tag{1.19}$$
$$x = \frac{-e}{q} \tag{1.20}$$

The same argument for the other bracket gives

$$x = \frac{-k}{r} \tag{1.21}$$

A simple program can then be written,

```
1  q = float(raw_input('enter q: '))
2  e = float(raw_input('enter e: '))
3  r = float(raw_input('enter r: '))
4  k = float(raw_input('enter k: '))
5
6  print "x=", -e / q
7  print "x=", -k / r
```

Alternatively, if you just got as far as comparing coefficients and putting them in the quadratic equation you'd get a more cumbersome (but just as correct),

```
q = float(raw_input("enter q: "))
e = float(raw_input("enter e: "))
r = float(raw_input("enter r: "))
k = float(raw_input("enter k: "))

print "x=", (-q*k - e*r + ((q*k + e*r)**2 - 4*q*r*e*k)**0.5) / (2*q*r)
print "x=", (-q*k - e*r - ((q*k + e*r)**2 - 4*q*r*e*k)**0.5) / (2*q*r)
```

2 Functions & Control

1. Take your program that finds the roots of the quadratic equation given a, b, and c, and write it as a function. *Hint: as the return keyword only returns one object, you will need to find a way to combine your two x values into one object*

Solution:

```
a = float(raw_input("enter a: "))
b = float(raw_input("enter b: "))
c = float(raw_input("enter c: "))

def find_roots(a, b, c):
    x1 = (-b + (b**2 - 4 * a * c)**0.5) / (2 * a)
    x2 = (-b - (b**2 - 4 * a * c)**0.5) / (2 * a)
    text1 = "\nThe quadratic equation y = {}x^2 + {}x + {} has solutions at\n".format(a, b, c)
    text2 = "x = {}\n".format(x1)
    text3 = "x = {}\n".format(x2)
    return text1 + text2 + text3
```

2. Write a function that asks for the user's age, and then gives them information about what they can or can't do at that age.

Solution:

```
def age_insulter(age)
    if age >= 65:
        return "Have you collected your pension yet today?"
    elif age >= 18:
        return "You can vote in UK general elections."
    elif age >= 17:
        return "You can apply for a provisional driving licence."
```

3. Write a function that returns the sum of the internal angles of a regular n-polygon.

Solution:

```
def polygon_angle(n):
    return 180 * (n - 2)
```

4. Write a function that insults the user based on their favourite subject.

Solution:

```
def subject_insulter(subject)
    if subject.lower() == "physics":
        return "A sensible choice. Rutherford would agree."
    elif subject.lower() == "chemistry":
        return "What is chemistry? It's just colour mixing."
    elif subject.lower() == "biology":
        return "Where animals are named in Latin, processes in Greek, and everything else in gibberish."
    elif subject.lower() == "maths":
        return "Here's your least favourite inequality: maths < physics."
    else:
        return "Well it's not quite physics is it?"
```

5. Write a function that returns the sum of the positive numbers up to n.

Solution:

```
def sum_to_n(n):
    if n == 0:
        return 0
    else:
        -return n + sum_to_n(n - 1)
```

6. Write a function that returns the nth Fibonacci number, where the Fibonacci sequence is defined as

$$F_n = F_{n-1} + F_{n-2} \tag{2.1}$$

with $F_0 = 0$ & $F_1 = 1$.

Solution:

```
def fib(n):
    if n == 1:
        return 1
    elif n == 0:
        return 0
    else:
        return fib(n - 1) + fib(n - 2)
```

7. Write a function that calculates the factorial of a number, where the factorial function is defined for positive integers as

$$n! = \begin{cases} n\,(n-1)! & \text{if } n \geq 2 \\ 1 & \text{otherwise} \end{cases} \tag{2.2}$$

Solution: This requires the use of *recursion*, which is the idea in computer science that a function can call itself. In this case, the factorial function calls itself on ever-decreasing numbers until it gets to 0.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

If `n=3` then when `factorial(3)` is called it returns `3 * factorial(2)`. The function `factorial(2)` then returns `2 * factorial(1)`, which returns `1`. So our function chain returns the result `3 * 2 * 1`, which is the definition of 3!

```
def factorial(n):
    # The factorial function is only defined for integers
    if n % 1 == 0:
        if n == 0:
            return 1
        else:
            return n * factorial(n-1)
    else:
        return ValueError
```

8. Write a function that tests the users ability to time a short period. The python module `datetime` has a function that gets the current computer time. You can import the module by using the line `import datetime as dt` and then using the function `dt.datetime.now()` to get the current time. You can add or subtract times in the usual way, and you can add or subtract intervals of time by using `dt.timedelta(seconds=10)`, shown here the time interval 10 seconds.

Solution:

```
import datetime as dt

def timing_game(time):
    raw_input(":>>>\tpress enter to start {} second timer".format(time))
    start = dt.datetime.now()

    raw_input(":>>>\tpress enter to end timer")
    end = dt.datetime.now()

    print
    print "Your time:", end - start

    if dt.timedelta(seconds=10) >= (end - start):
        print "You were out by -{diff}.".format(diff=dt.timedelta(seconds=10) - (end - start))
    else:
        print "You were out by +{diff}.".format(diff=(end - start) - dt.timedelta(seconds=10))

    raw_input("press any key to exit")
```

3 Loops & Lists

1. Write a function that takes as its argument a list of values and returns a sum of that list.
2. Write a function that counts down from 10.
3. Define a function `maximum()` that takes two numbers as arguments and returns the largest of them.
4. Modify your function that returns the sum of the positive numbers up to n so that it uses a loop instead of recursion.
5. Write a function that prints out a table of positive integers, their squares, and cubes. Your table should be nicely formatted to allow the display of at least the first 10 integers.
6. Write a function that takes a list of values as its argument and returns the sum of the even elements minus the sum of the odd elements.