Motivating Examples by Nick Maclaren

As a warm-up, we will consider a number of surprising example, out of which many are due to Nick Maclaren of Cambridge University. He uses them in his class “Numerical Programming in Python”.

EXAMPLE 0.1. Conversion to floating point numbers introduces errors:

```python
print 1.1+2.2, (1.1+2.2) == 3.3
print "%.16f %.16f %.16f" % (1.1, 2.2, 1.1+2.2)
```

Listing 1. Motivating example from Python Decimal documentation

EXAMPLE 0.2. Addition results in errors and \((a + x)x\) need not be \(a\):

```python
x = 0.001
y = (1.0+x)-1.0
print x, y, x == y
print "%.16f %.16f" % (x,y)
```

EXAMPLE 0.3. You can increase the precision by considering module decimal:

```python
from decimal import Decimal
a = Decimal('0.1')
b = Decimal('0.2')
c = a + b
```

EXAMPLE 0.4. \(a + a + a\) is not guaranteed to be \(3.0 \ast a\):

```python
x = 1.0/6.0
y = x+x+x+x+x+x
print y, y == 1.0
print "%.18f %.18f" % (x, y)
```

EXAMPLE 0.5. \(a\) and \(1.0/a\) are not guaranteed to \(a(1.0/a) = 1\):

```python
from math import e
x = e/11.0
y = 1.0/x
z = 1.0/y
print x == z
print "%.18f %.18f %.18f" % (x,y,z)
```
Example 0.6. The range for doubles is from $2.2 \cdot 10^{-308}$ to $1.8 \cdot 10^{308}$:

```python
x = 1.0e-20
y = 5.0e-324
print 1.0+x == 1.0, y/2.0
print "%16e %.16f %.16f" % (x,y/2,2*(y/2.0))
```

Example 0.7. $a > b$ and $c > d$ need not mean $a + c > b + d$:

```python
a = 0.75+1.0e-16
b = 0.75
c = 0.5
d = 0.5-1.0e-16
print a > b, c > d, a+c > b+d
print "%16f %.16f %.16f %.16f" % (a,b,c,d)
print "%16f %.16f" % (a+c,b+d)
```
EXERCISE 0.8 (2 points). Consider the following simulator of a Turing machine (TM):

```python
def turing(code, tape, initPos = 0, initState = "1"):  
    position = initPos  
    state = initState  
    while state != "halt":  
        print state, ": position", position, ", in", tape  
        symbol = tape[position]  
        (symbol, direction, state) = code[state][symbol]  
        if symbol != "noWrite": tape[position] = symbol  
        position += direction
```

Implement a TM, which multiplies two integers, which are encoded on the tape in unary and delimited by “blank” on both ends and between the numbers. Do not replace the numbers, but append the result after yet another blank.

Hint: Unary encoding means the number of a occurrences of a particular symbols (e.g., “1”) is equal to the number (e.g., “11111” stands for 5).

EXERCISE 0.9 (2 points). Consider the following simulator of a BSS machine:

```python
def bss(code, outgoing, initValues = [0.0], initNode = "1"):  
    regs = initValues  
    node = initNode  
    counter = 0  
    while outgoing.has_key(node) and counter < 100:  
        counter += 1  
        print counter, ":", node, code[node]  
        nodeType = code[node][0]  
        if nodeType == "branch":  
            branchType = code[node][1]  
            (i, j) = code[node][2:]  
            (left, right) = outgoing[node]  
            if branchType == "<=" and regs[i] <= regs[j]: node = left  
            if branchType == "==" and regs[i] == regs[j]: node = left  
            if branchType == "=>=" and regs[i] >= regs[j]: node = left  
            node = right  
        else:  
            (i, j) = code[node][1:]  
            if nodeType == "assign": regs[i] = j  
            elif nodeType == "copy": regs[i] = regs[j]  
            elif nodeType == "add": regs[0] = regs[1] + regs[2]  
            elif nodeType == "subtract": regs[0] = regs[1] - regs[2]  
            elif nodeType == "multiply": regs[0] = regs[1] * regs[2]
```
elif nodeType == "divide": regs[0] = regs[1] / regs[2]

else: print "Unknown instruction"

node = outgoing[node]

Implement a BSS for it, which uses Newton’s method for computing the square root of a real number \( S \) provided in one of the input registers.

Hint: one can apply Newton’s method to \( x^2 - S = 0 \), but simpler methods first finds \( 1/\sqrt{S} \) by solving \( (1/x^2) - S = 0 \) and subsequently obtain \( \sqrt{S} = S \cdot (1/\sqrt{S}) \). A single iteration of solving \( (1/x^2) - S = 0 \) can take the form of \( x_{n+1} = \frac{x_n^2}{2} \cdot (3 - S \cdot x_n^2) \).

**Exercise 0.10** (1 point). Expand the following code by Christian Bauckhage to create a cartoon by varying parameters of the Julia Set. Try plotting it in gray scale.

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm

def Julia(f, zmin, zmax, m, n, tmax=256):
    xs = np.linspace(zmin.real, zmax.real, n)
    ys = np.linspace(zmin.imag, zmax.imag, m)
    X, Y = np.meshgrid(xs, ys)
    Z = X + 1j * Y
    J = np.ones(Z.shape) * tmax
    for t in xrange(tmax):
        mask = np.abs(Z) <= 2.
        Z[mask] = f(Z[mask])
        J[-mask] -= 1
    return J

zmin = -1.3 - 1j * 1.3
zmax = 1.3 + 1j * 1.3
J = Julia(lambda z: z**2 -.065+.66j, zmin, zmax, m=1024, n=1024)
plt.imsave("julia.png", J, cmap=cm.jet, origin="lower")
```

4