Three procedures for computing sums

```python
def sumint(low, high):
    s = 0
    x = low
    while x <= high:
        s = s + x
        x = x + 1
    return s

def sumsquares(low, high):
    s = 0
    x = low
    while x <= high:
        s = s + x**2
        x = x + 1
    return s

def piSum(low, high):
    s = 0
    x = low
    while x < high:
        s = s + 1.0 / x**2
        x = x + 2
    return s
```

Approximation to $\pi^2/8$

```python
def piSum(low, high):
    s = 0
    x = low
    while x < high:
        s = s + 1.0 / x**2
        x = x + 2
    return s
```

The general idea of summation, expressed as a procedure that captures the common pattern: $\sum_{a}^{b} f$: 

```python
def summation(low, high, f, next):
    s = 0
    x = low
    while x <= high:
        s = s + f(x)
        x = next(x)
    return s
```
The `sumint` procedure, expressed as a general sum

```python
def sumint(low, high):
    def identity(x):
        return x
    def add1(x):
        return x + 1
    return summation(low, high, identity, add1)
```

The same three sums, expressed in terms of the general idea of summation, using `lambda` to avoid having to name the internal procedures:

```python
def sumsquares(low, high):
    return summation(
        low,
        high,
        lambda x: x**2,
        lambda x: x + 1
    )

def sumsquares(low, high):
    return summation(
        low,
        high,
        lambda x: x**2,
        lambda x: x + 1
    )

def piSum(low, high):
    return summation(low,
                     high,
                     lambda x: 1.0 / x**2,
                     lambda x: x + 2)
```

Expressing a general method of finding a fixed point of a function `f`:

```python
def fixedPoint(f, firstGuess):
    def close(g1, g2):
        return abs(g1 - g2) < .0001
    def iter(guess, next):
        while True:
            if close(guess, next):
                return next
            else:
                guess = next
                next = f(next)
        return iter(firstGuess, f(firstGuess))
```
Then we can compute square roots as fixed points:

```python
def sqrt(x):
    def average(a,b): return (a+b)/2.0
    return fixedPoint(lambda g: average(g,x/g),1.0)
```

Four procedures for computing the sum of \( f(x) = x\sqrt{x} \) for all the numbers in a list. They all do the same computation, but are expressed differently.

```python
def sumf1(p):
    result = 0
    i = 0
    while i < len(p):
        result = result + p[i]*sqrt(p[i])
        i = i + 1
    return result

def sumf2(p):
    result = 0
    for x in p:
        result = result + x*sqrt(x)
    return result

def sumf3(p):
    return reduce(add, [x*sqrt(x) for x in p])

def sumf4(p):
    return reduce(add, map(lambda x: x*sqrt(x),p))
```

Computing derivatives: Given a function \( f \), the derivative \( Df \) is another function. Therefore \( D \) itself is a function whose value is a function:

```python
def deriv(f):
    dx=0.0001
    return lambda x: (f(x+dx)-f(x))/dx
We can write this equivalently, without using \texttt{lambda}:

\begin{verbatim}
def deriv(f):
    dx=0.0001
    def d(x):
        return (f(x+dx)-f(x))/dx
    return d
\end{verbatim}

In either case, if we apply \texttt{deriv} to a procedure, the result is another procedure, that we can then apply to a number, e.g.,

\begin{verbatim}
>>> deriv(square)(10)
\end{verbatim}

This returns 20 (approximately) because the derivative of \( x \mapsto x^2 \) is \( x \mapsto 2x \).

Once we can express derivative, we can express Newton’s method:

\begin{verbatim}
def newtonsMethod(f,firstGuess):
    return fixedPoint(
        lambda x: x - f(x)/deriv(f)(x),
        firstGuess)
\end{verbatim}

and we can express computing square roots as an application of Newton’s method:

\begin{verbatim}
def sqrt(x):
    return newtonsMethod(
        lambda y: y**2 - x,
        1.0)
\end{verbatim}

The general method of iterative improvement, expressed as a procedure:

\begin{verbatim}
def iterativeImprove(goodEnough,improve,start):
    result = start
    while not goodEnough(result):
        result = improve(result)
    return result
\end{verbatim}

\textbf{Rights and privileges of first-class citizens in programming languages} \hspace{1em} (Christopher Strachey)

- May be named by variables
- May be passed as arguments to procedures
- May be returned as results of procedures
- May be included in data structures