Procedural vs functional programming

Basic examples with *Mathematica*

Hugo Touchette
School of Mathematical Sciences
Queen Mary, University of London

Programming Seminar

Date: 27 October 2010

Ideas

- Act on lists or structures
- Avoid loops
- Use pure functions
- Think global/operators/composition (vs local/sequential)
- Don't care too much about space/memory

Basic example

Problem:
Sum integers from 1 to 1 000 000

Direct answer

In[15]:=
```
Sum[i, {i, 1 000 000}]
```

Out[15]= 500 000 500 000

Procedural

In[16]:=
```
s = 0;
Do[
  s = s + i,
  {i, 1 000 000}];
```

In[18]:=
```
s
```

Out[18]= 500 000 500 000

Functional I

In[19]:=
```
integerlist = Table[i, {i, 1 000 000}];
```

In[20]:=
```
Fold[#1 + #2 &, 0, integerlist]
```

Out[20]= 500 000 500 000

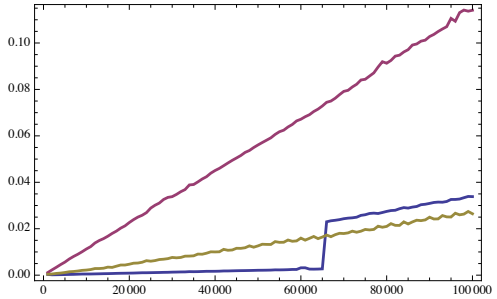
Functional II

In[21]:=
```
Plus @@ integerlist
```

Out[21]= 500 000 500 000

Time comparison

```
test1[n_] := Module[{i}, Sum[i, {i, n}]]
test2[n_] := Module[{s, i}, s = 0; Do[s = s + i, {i, n}]; s]
test3[n_] := Plus @@ Table[i, {i, n}]

DiscretePlot[{Timing[test1[n]][[1]], Timing[test2[n]][[1]], Timing[test3[n]][[1]]},
 {n, 1000, 100 000, 1000}, Filling → None, Joined → True, Frame → True, PlotStyle → Thick]
```



Everything is an expression

In[22]:=
```
1 + 2 + 3 + 4 // Hold // FullForm
```

Out[22]//FullForm=
```
Hold[Plus[1, 2, 3, 4]]
```


In[23]:=
```
{1, 2, 3, 4} // FullForm
```

Out[23]//FullForm=
```
List[1, 2, 3, 4]
```

In[24]:=
```
Plus @@ {1, 2, 3, 4}
```

Out[24]= 10

In[25]:=
```
Apply[Plus, {1, 2, 3, 4}]
```

Out[25]= 10

In[26]:=
```
Times @@ {1, 2, 3, 4}
```

Out[26]= 24

In[27]:=
```
Apply[Times, {1, 2, 3, 4}]
```

Out[27]= 24

In[28]:=
```
f[x] /. f → g
```

Out[28]= g[x]

In[29]:=
```
MyMean[l_] := Plus @@ l / Length[l]
```

In[30]:=
```
MyMean[{1, 2, 3, 4}]
```

Out[30]= $\frac{5}{2}$

Pure functions

- Pure functions (*Mathematica*)
- Anonymous functions
- Lambda functions (Lambda calculus, Lisp, Python)

In[31]:=
```
#^2 &[c]
```

Out[31]= c^2

In[32]:=
```
Function[u, u^2][c]
```

Out[32]= c^2

In[33]:=
```
#^2 & /@ {a, b, c}
```

Out[33]= $\{a^2, b^2, c^2\}$

In[34]:=
```
Map[#^2 &, {a, b, c}]
```

Out[34]= $\{a^2, b^2, c^2\}$

In[35]:=
```
f /@ (a + b + c)
```

Out[35]= f[a] + f[b] + f[c]

In[36]:=
```
(#[[1]] + #[[2]] &) /@ {{a, b}, {c, d}}
```

Out[36]= {a + b, c + d}

In[37]:=
```
Nest[#^c &, x, 3]
```

Out[37]= $\left(\left(x^c\right)^c\right)^c$

Example: Newton-Raphson iteration

```
In[38]:= NewtonZero[f_, x0_] := FixedPoint[# - f[#] / f'[#] &, x0]
In[39]:= NewtonZero[BesselJ[2, #] &, 5.0]
Out[39]= 5.13562
```

■ Another example

```
In[40]:= EmitSound[Sound[SoundNote[#, .1, "Percussion"] & /@ Range[-32, 35]]]
```

Composition and pipeline

■ Prefix

```
In[41]:= f@x
Out[41]= f[x]
In[42]:= g@f@x
Out[42]= g[f[x]]
```

■ Postfix (pipeline)

```
In[43]:= x // f
Out[43]= f[x]
In[44]:= x // f // g
Out[44]= g[f[x]]
```

■ Composition

```
In[45]:= Nest[f, x, 4]
Out[45]= f[f[f[f[x]]]]
In[46]:= NestList[f, x, 4]
Out[46]= {x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]]]}
In[47]:= Nest[1 + 1 / # &, x, 3]
```

$$\text{Out[47]}= 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{x}}}$$

Listable functions

```
In[48]:= data = Table[RandomReal[], {10^5}];
```

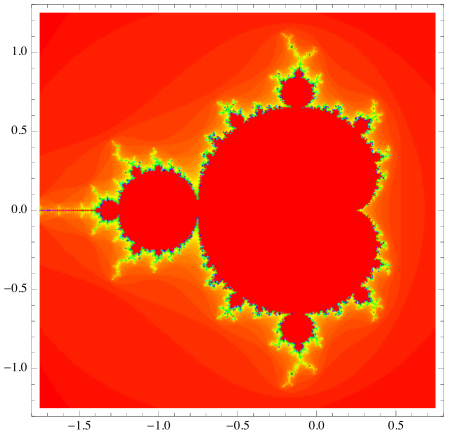


```
In[49]:= Exp /@ data; // Timing
Out[49]= {0.01435, Null}
In[50]:= Exp[data]; // Timing
Out[50]= {0.003079, Null}
In[51]:= ans = data;
         k = Length[data];
         While[k > 0,
           ans[[k]] = Exp[data[[k]]];
           k--;
         ]; // Timing
Out[53]= {0.463547, Null}
```

Application: Mandelbrot set

■ Procedural

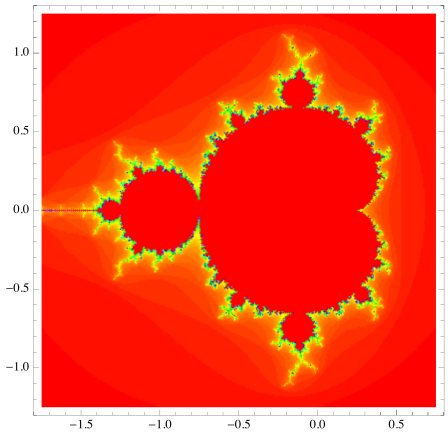
```
Mandelbrotset0[x_, y_] := Module[{z, t = 0, c = x + I y},
  z = c;
  While[(Abs[z] < 2.0) && (t < 100),
   z = z^2 + c;
   t++];
  t
 ];
DensityPlot[Mandelbrotset0[x, y], {x, -1.75, 0.75},
 {y, -1.25, 1.25}, PlotPoints → 100, Mesh → False, ColorFunction → Hue]
```



■ Functional

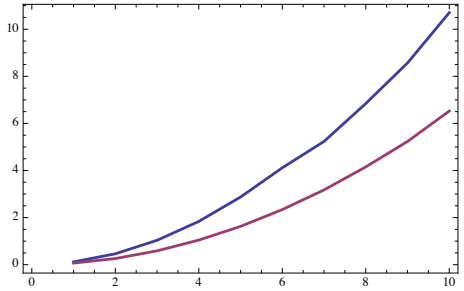
```
Mandelbrotset2[c_, nmax_] :=
 Length[FixedPointList[#^2 + c &, c, nmax, SameTest → (Abs[#2] > 2.0 &)]]
```

```
DensityPlot[Mandelbrotset2[x + y I, 100], {x, -1.75, 0.75},
 {y, -1.25, 1.25}, PlotPoints → 100, Mesh → False, ColorFunction → Hue]
```



■ Time comparison

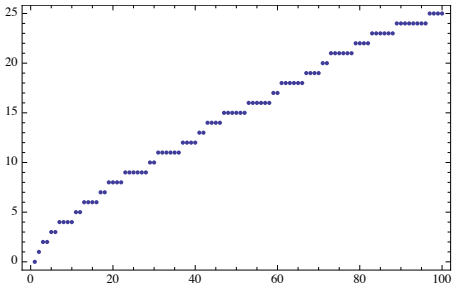
```
timingtable1 = Table[
   Timing[Table[Mandelbrotset0[x, y], {x, -1.75, 0.75, 1 / n}, {y, -1.25, 1.25, 1 / n}]][[1]],
   {n, 10, 100, 10}];

timingtable2 = Table[Timing[Table[Mandelbrotset2[x + I y, 100],
      {x, -1.75, 0.75, 1 / n}, {y, -1.25, 1.25, 1 / n}]][[1]], {n, 10, 100, 10}];

ListPlot[{timingtable1, timingtable2}, Joined → True, Frame → True, PlotStyle → Thick]
```



Prime counting function

■ Procedural

```
πproc[n_] := Module[{cnt, i},
   cnt = 0;
   Do[
    If[PrimeQ[i], cnt++],
    {i, 2, n}];
   cnt
  ];

ListPlot[Table[{i, πproc[i]}, {i, 1, 100}]]
```



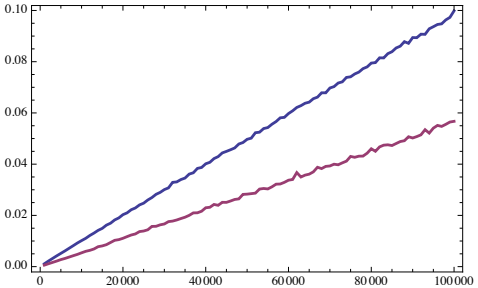
■ Functional

1. List all the positive integers up to *n*
2. Extract the primes
3. Count what's extracted

```
πfunc[n_] := Length[Select[Table[i, {i, n}], PrimeQ]]
```

■ Time comparison

```
DiscretePlot[{Timing[πproc[n]][[1]], Timing[πfunc[n]][[1]]},
 {n, 1000, 100 000, 1000}, Filling → None, Joined → True, Frame → True, PlotStyle → Thick]
```



Perfect numbers

A number n is **perfect** if it is equal to the sum of its proper divisors (all divisors except n) or, equivalently, if the sum of all its divisors (including n) is equal to $2\,n$.

For example, 6 is a perfect number since its proper factors, 1, 2, 3 sum to $1 + 2 + 3 = 6$.

In[54]:=
```
PerfectNumberQ[n_] := 2 n === Plus @@ Divisors[n]
```

In[56]:=
```
PerfectNumberQ[12]
```

Out[56]=
```
False
```

▪ Procedural

```
With[{nmax = 1 000 000},
  l = {};
  Do[
   If[PerfectNumberQ[i], AppendTo[l, i]],
   {i, nmax}];
  l
 ] // Timing
```

```
{13.5462, {6, 28, 496, 8128}}
```

▪ Functional

```
With[{nmax = 1 000 000},
  Select[Table[i, {i, 1, nmax}], PerfectNumberQ]
 ] // Timing
```

```
{12.8476, {6, 28, 496, 8128}}
```

Prime sums

$$S_n = \frac{1}{n}\sum_{i=1}^{n}\pi_i$$

Generate the list $\{n, S_n\}_{n=1}^{m}$

▪ Procedural

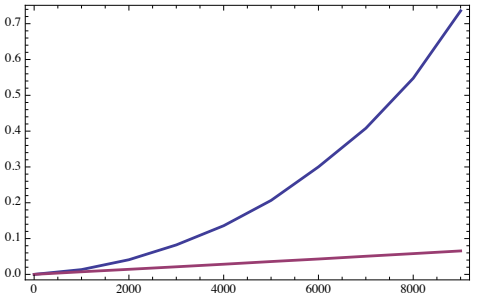
```
primesumproc[n_] := Module[{i, ps, res},
  ps = 0;
  res = {};
  Do[
   ps = ps + Prime[i];
   AppendTo[res, {i, ps / i}],
   {i, 1, n}];
  res
 ]
```

▪ Functional

```
primesumfunc[n_] := NestList[
  {#[[1]] + 1, (#[[2]] #[[1]] + Prime[#[[1]] + 1]) / (#[[1]] + 1)} &,
  {1, 2}, n - 1
 ]
```


Time comparison

```
With[{n = 10 000, dn = 1000},
 DiscretePlot[{Timing[primesumproc[i]][[1]], Timing[primesumfunc[i]][[1]]},
  {i, 1, n, dn}, Filling → None, Joined → True, Frame → True, PlotStyle → Thick]
]
```



1, 3, 5, 5 question

Interview question:
What is the combination of the numbers 1, 3, 5 and 5 with any of the arithmetic operations (+, -, *, /) that yields 16?

▪ Include

```
In[1]:= ListDelete[list_, elem_] := Delete[list, Position[list, elem][[1, 1]]]

In[2]:= ListOperation[l_] :=
         Module[{res, i}, res = l[[1]]; Do[res = res ~ l[[i]] ~ l[[i + 1]], {i, 2, Length[l], 2}]; res]
```

▪ Procedural

```
In[3]:= numset = {1, 3, 5, 5};

In[4]:= opset = {Plus, Subtract, Times, Divide};

In[57]:= Catch[
          Do[
           numset1 = ListDelete[numset, n1];
           Do[
            numset2 = ListDelete[numset1, n2];
            Do[
             opset1 = ListDelete[opset, op1];
             res1 = n1 ~ op1 ~ n2;
             Do[
              numset3 = ListDelete[numset2, n3];
              Do[
               opset2 = ListDelete[opset1, op2];
               res2 = res1 ~ op2 ~ n3;
               Do[
                n4 = numset3[[1]];
                res3 = res2 ~ op3 ~ n4;
                If[res3 == 16, Throw[{n1, op1, n2, op2, n3, op3, n4}]], {op3, opset2}],
              {op2, opset1}],
             {n3, numset2}],
            {op1, opset}],
           {n2, numset1}],
          {n1, numset}]
         ]

Out[57]= {1, Divide, 5, Plus, 3, Times, 5}
```

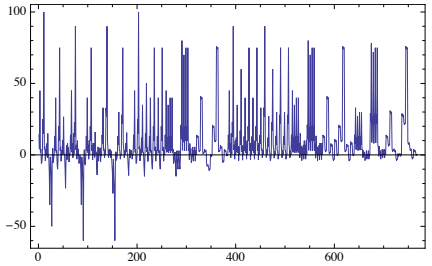
▪ Functional

```
In[58]:= listperm = Permutations[numset]

Out[58]= {{1, 3, 5, 5}, {1, 5, 3, 5}, {1, 5, 5, 3}, {3, 1, 5, 5}, {3, 5, 1, 5}, {3, 5, 5, 1},
          {5, 1, 3, 5}, {5, 1, 5, 3}, {5, 3, 1, 5}, {5, 3, 5, 1}, {5, 5, 1, 3}, {5, 5, 3, 1}}
```



```
In[59]:= listop = Flatten[Table[{i, j, k}, {i, opset}, {j, opset}, {k, opset}], 2]

Out[59]= {{Plus, Plus, Plus}, {Plus, Plus, Subtract}, {Plus, Plus, Times}, {Plus, Plus, Divide},
          {Plus, Subtract, Plus}, {Plus, Subtract, Subtract}, {Plus, Subtract, Times},
          {Plus, Subtract, Divide}, {Plus, Times, Plus}, {Plus, Times, Subtract},
          {Plus, Times, Times}, {Plus, Times, Divide}, {Plus, Divide, Plus},
          {Plus, Divide, Subtract}, {Plus, Divide, Times}, {Plus, Divide, Divide},
          {Subtract, Plus, Plus}, {Subtract, Plus, Subtract}, {Subtract, Plus, Times},
          {Subtract, Plus, Divide}, {Subtract, Subtract, Plus}, {Subtract, Subtract, Subtract},
          {Subtract, Subtract, Times}, {Subtract, Subtract, Divide}, {Subtract, Times, Plus},
          {Subtract, Times, Subtract}, {Subtract, Times, Times}, {Subtract, Times, Divide},
          {Subtract, Divide, Plus}, {Subtract, Divide, Subtract}, {Subtract, Divide, Times},
          {Subtract, Divide, Divide}, {Times, Plus, Plus}, {Times, Plus, Subtract},
          {Times, Plus, Times}, {Times, Plus, Divide}, {Times, Subtract, Plus},
          {Times, Subtract, Subtract}, {Times, Subtract, Times}, {Times, Subtract, Divide},
          {Times, Times, Plus}, {Times, Times, Subtract}, {Times, Times, Times},
          {Times, Times, Divide}, {Times, Divide, Plus}, {Times, Divide, Subtract},
          {Times, Divide, Times}, {Times, Divide, Divide}, {Divide, Plus, Plus},
          {Divide, Plus, Subtract}, {Divide, Plus, Times}, {Divide, Plus, Divide},
          {Divide, Subtract, Plus}, {Divide, Subtract, Subtract}, {Divide, Subtract, Times},
          {Divide, Subtract, Divide}, {Divide, Times, Plus}, {Divide, Times, Subtract},
          {Divide, Times, Times}, {Divide, Times, Divide}, {Divide, Divide, Plus},
          {Divide, Divide, Subtract}, {Divide, Divide, Times}, {Divide, Divide, Divide}}

In[60]:= Riffle[listperm[[1]], listop[[1]]]

Out[60]= {1, Plus, 3, Plus, 5, Plus, 5}

In[61]:= allcaseslist = Flatten[
           Table[
            Riffle[listperm[[i]], listop[[j]]],
            {i, Length[listperm]},
            {j, Length[listop]}],
           1
          ];

In[62]:= resultlist = ListOperation /@ allcaseslist;

In[66]:= ListPlot[resultlist, Joined → True, PlotRange → All]
```

Out[66]=




```
In[64]:= winningpos = Position[ListOperation /@ allcaseslist, 16]

Out[64]= {{115}}

In[65]:= Extract[allcaseslist, winningpos]

Out[65]= {{1, Divide, 5, Plus, 3, Times, 5}}
```

Conclusions

Reform your sequential brain!

Think lists

- Think listable functions
- Think global operations
- Avoid loops
- Avoid counters
- Think functional programming