Excursions in Computing Science: Week 12. Memory and Programming Language: Recursion and Instantiation

T. H. Merrett*
McGill University, Montreal, Canada

August 26, 2009

I. Prefatory Notes

A. Recursion

1. Define "ancestor" in terms of "parent".

2. Precedence without parentheses

We'd like to parse `ab + cd` in the conventional way: `(ab) + (cd)`.

We can say `ab + cd` is an *expression* made up of *terms*.
"Made up" means combined by + (**or**).

Likewise, terms are made up of *identifiers* (letters), i.e., combined by adjacency, or sometimes • (**and**).

Grammar

```
<expression>::= <term> | <term> + <expression>
<term>::= <letter> | <letter><term>
```

These *recursive* definitions allow us to have indefinite numbers of terms with indefinite numbers of letters.

E.g., `a + bc + def + gh`

Let's see if we can use these recursive guidelines to write a program to recognize such an expression and put parentheses around the terms.

```
parsexp('ab+cd')

ans =

((ab)+(cd))
```

1. `parsexp` puts out a '(', calls `expression`, and puts out a ')'.

 The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students and their funding agencies.

2. `expression` puts out a '(', calls `term`, puts out a ')', calls `plussign` and, if `plussign` found a plus in the input, calls *itself* recursively to get the next expression.

3. `plussign` puts out '+' and reports success if '+' is indeed the next character of the input; otherwise it reports failure.

4. `term` calls `letter` and, if `letter` found a letter in the input, calls *itself* recursively to get the next term.

5. `letter` puts out the letter and reports success if the next character of the input is indeed a letter; otherwise it reports failure.

Here is `parsexp`. Note that MATLAB obliges us to keep explicit account of the input (`code`), our current input position (`inpos`), and the output (`parsed`) and our current output position (`outpos`).

```
% function parsed = parsexp(code)
% THM 060829
% uses expression.m
function parsed = parsexp(code)
parsed(1) = '(';
[parsed,inpos,outpos] = expression(code,parsed,1,2);
parsed(outpos) = ')';
```

Here is `expression`. I'll leave `plussign, term` and `letter` as exercises.

```
% function [parsed,inpos,outpos] = expression(code,parsed,inpos,outpos)
% THM 060829
% used by parsexp.m; uses term.m, plussign.m; term.m uses letter.m
function [parsed,inpos,outpos] = expression(code,parsed,inpos,outpos)
parsed(outpos) = '('; outpos = outpos + 1;
[parsed,inpos,outpos] = term(code,parsed,inpos,outpos);
parsed(outpos) = ')'; outpos = outpos + 1;
[succ,parsed,inpos,outpos] = plussign(code,parsed,inpos,outpos);
if succ
 [parsed,inpos,outpos] = expression(code,parsed,inpos,outpos);
end
```

We can see that recursive thinking makes the process very much easier than without recursion.

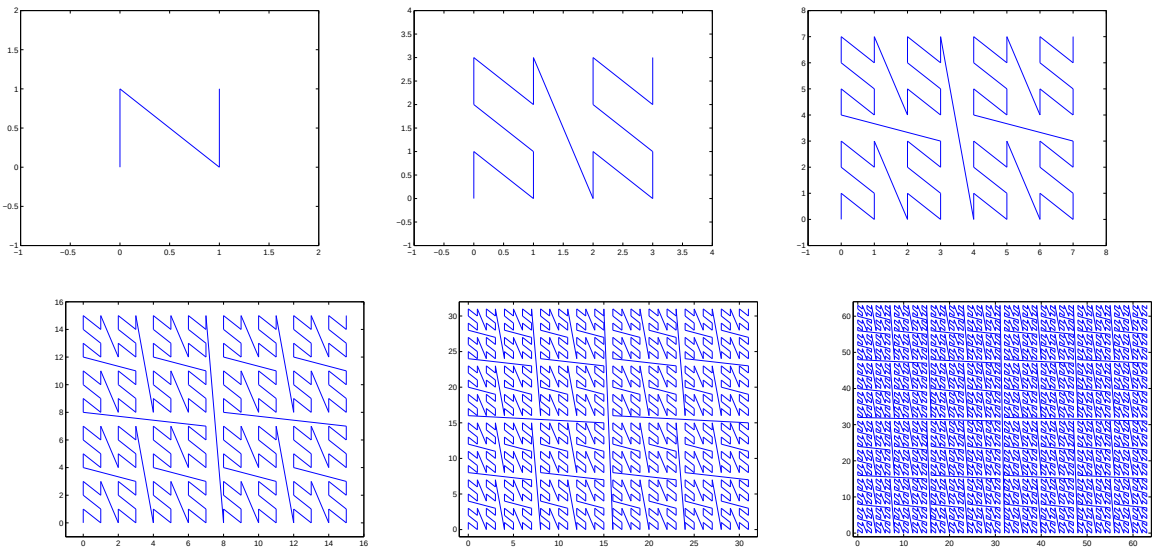
3. Fractals

A significant benefit of recursion is in computing an *exponential number* of things.

`Expression` and `term` only called themselves once, but if a function calls itself twice and does this recursively to depth n then we can get 2^n invocations of the function.

We don't often *want* an exponential number of invocations because it is very expensive (see the Excursion this week about the chessboard), but one attractive application is drawing self-similar figures (*fractals*).

Here is the "Peano curve", named after the French mathematician who published it in 1880 [Pea90]. I have run the program to depths 1–6.



1. `peano(n)` sets up the arrays that MATLAB needs to represent the final drawing, calls `peanoStep(n)`, and then uses MATLAB's `quiver()` and `axis()` functions to produce the drawing.

2. `peanoStep(n)` calls `peanoStep(n−1)` four times and also draws three lines, one in between each invocation: first vertically up then diagonally down and rightwards then vertically up again; the length of the lines depends on the current level, n.

Here is `peanoStep()`. Remember again that MATLAB obliges us to do all the housekeeping explicitly, including keeping track of the "screen" represented by the arrays X, Y, U and V.

```
% function [X,Y,U,V,j,k] = peanoStep(n,X,Y,U,V,j,k)
% THM 060829
% called from peano(n); uses draw.m
function [X,Y,U,V,j,k] = peanoStep(n,X,Y,U,V,j,k)
if n>0
  [X,Y,U,V,j,k] = peanoStep(n-1,X,Y,U,V,j,k);
  x =1-2^(n-1); y = 1; draw
  [X,Y,U,V,j,k] = peanoStep(n-1,X,Y,U,V,j,k);
  x = 1; y =1-2^n; draw
  [X,Y,U,V,j,k] = peanoStep(n-1,X,Y,U,V,j,k);
  x =1-2^(n-1); y = 1; draw
  [X,Y,U,V,j,k] = peanoStep(n-1,X,Y,U,V,j,k);
end
```

(A nice thing about recursive programming is that as you improve your code it gets *smaller*. My original version of `peanoStep()` had a stopping condition for $n = 1$. Why is this neither needed nor very good?)

(When the Peano curve is drawn in any number of dimensions (including 2-D as a special case), it is called "Z-order". It can actually be drawn, to fixed depth, without recursion by interleaving (or "shuffling") the bits that represent the coordinates of the grid points being connected by the Z-order.)

4. Mathematical induction.

A good way to think about recursion is as a form of proof by mathematical induction. This

requires an *induction step*, which is the recursive call, and a *starting step*, which becomes the *stopping condition* in a recursive program.

Typically, mathematical induction is used to prove something is true for all integers, n, by showing

1. the "something" is true for $n = 1$;
2. if the "something" is true for $n - 1$, then it is true for n.

This is related to linear (non-exponential) recursion.

Another linear recursion is used to find the greatest common divisor of two integers (and the corresponding mathematical induction is proof that the recursion is correct):

```
function g = gcd(x,y)
  if y==0, g=x;                  %stopping condition
  else g = gcd(y,rem(x,y));      %recursion/induction step
```

Try this on $x = 38$, $y = 14$. Try it with paper and pencil: the sequence of calls is

step	0	1	2	3	4
x	38	14	10	4	2
y	14	10	4	2	0

We saw in week 10 that x and y have the same greatest common divisor (gcd) as y and x mod y. This is the induction step in a mathematical proof that the `gcd` function is correct, and it is the recusive step in the function. (The MATLAB function `rem(x,y)` and x mod y do the same thing. They give the remainder when integer x is divided by integer y.)

Furthermore, the recursion reduces the sizes of the parameters so they get smaller and smaller. Persuade yourself that y must eventually be 0, so that the x that made it so is an exact divisor of both the preceding x and y ... and so of the original x and y.

This was an inductive argument showing that the `gcd` program works. It is called Euclid's algorithm.

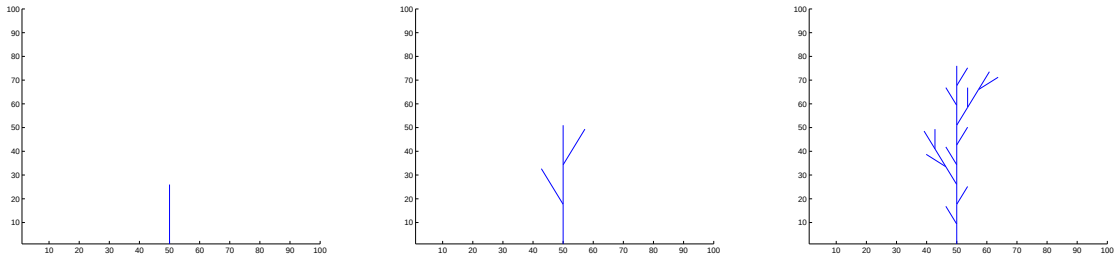
5. MATLAB syntax is not particularly elegant for recursion, and we can do better in other languages.

```
function gcd(x,y) is
  if y==0 then x
  else gcd(y,x mod y)
```

A programming language which supports only *expressions*, without needing *assignment statements*, is an example of *functional programming*. LISP was the first thorough example of this.

6. L-systems

Botany provides many illustrations of recursion: a branch produces sub-branches which produce sub-sub-branches, and so on.



Such botanical structures are also *self-similar*. How many times is the second "tree" drawn above found in the third? How does each of these correspond to a straight line in the second? Can you see what will happen next?

The basic pattern can be captured symbolically.

`B[+B]B[−B]B`

where

`B`: draw a line of given length
`+`: turn left by a given angle
−: turn right by a given angle
`[`: store current position and heading
`]`: retrieve stored position and heading

(`[` and `]` are *stack* operations, so several (position, heading) states may be stored, to be retrieved in the inverse order.)

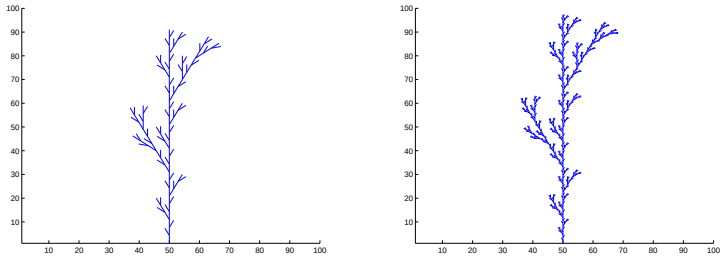
Now for the recursion. To make a self-similar version to the next degree of resolution, turn the symbolic string into a *rewriting rule*

$$\texttt{B} \rightarrow \texttt{B[+B]B[−B]B}$$

This has the effect of replacing each `B` in the string by the string itself.

Applied an indefinite number of times, the rewriting rule "grows" a structure of arbitrary complexity.

Here are the next two steps following the three shown above.



MATLAB can implement this, although in a limited way. Using `quiver` to draw the final picture, we can build up the picture recursively in a way which follows the symbolic string.

1. `branchOL1(max,n)` initializes the size of the picture, given by `max`, initializes arrays needed by the stack, and then calls `branchOL1Step(n,..)`.

2. `branchOL1Step(n,..)` calls `branchOL1Step(n−1,..)` five times, corresponding to the five occurences of `B` in the rewriting rule, or, if $n = 0$, directly calls `forward()` to draw the straight line represented by the lowest level of `B`. `branchOL1Step()` uses `pushBOLstate()` and `popBOLstate()` to do the stack operations where indicated by [and], respectively.

Here is `branchOL1Step()`. (It must explicitly pass as parameters and return as results all the global variables needed for the current position and heading, and for the stacked positions and headings.)

```
% function [x,y,h,sX,sY,sH,sNext] =
   branchOL1Step(n,step,delta,x,y,h,sX,sY,sH,sNext)
% THM 061023 in file: branchOL1Step.m
% called by branchOL1.m; uses forward.m popBOLstate.m, pushBOLstate.m
function [x,y,h,sX,sY,sH,sNext] =
   branchOL1Step(n,step,delta,x,y,h,sX,sY,sH,sNext)
if n==0, [U,V,x,y] = forward(step,U,V,x,y,h); else
```

```
  [x,y,h,sX,sY,sH,sNext] = branchOL1Step(n-1,step,delta,x,y,h,sX,sY,sH,sNext);%B
  [sX,sY,sH,sNext] = pushBOLstate(x,y,h,sX,sY,sH,sNext); %[ PUSH
  h = h + delta; %+ left(delta)
  [x,y,h,sX,sY,sH,sNext] = branchOL1Step(n-1,step,delta,x,y,h,sX,sY,sH,sNext);%B
  [x,y,h,sX,sY,sH,sNext] = popBOLstate(sX,sY,sH,sNext); %] POP
  [x,y,h,sX,sY,sH,sNext] = branchOL1Step(n-1,step,delta,x,y,h,sX,sY,sH,sNext);%B
  [sX,sY,sH,sNext] = pushBOLstate(x,y,h,sX,sY,sH,sNext); %[ PUSH
  h = h - delta; %- left(-delta)
  [x,y,h,sX,sY,sH,sNext] = branchOL1Step(n-1,step,delta,x,y,h,sX,sY,sH,sNext);%B
  [x,y,h,sX,sY,sH,sNext] = popBOLstate(sX,sY,sH,sNext); %] POP
  [x,y,h,sX,sY,sH,sNext] = branchOL1Step(n-1,step,delta,x,y,h,sX,sY,sH,sNext);%B
end
```

B. Instantiation

Comparing recursive with nonrecursive code, we see that the parameters of the recursive function provide a kind of *workspace*.

Recursive

```
function gcd(x,y) is
  if y==0 then x
  else gcd(y,x mod y)
```

Iterative

```
function gcd(x,y)
  while(y>0)
  { y' = x mod y;
    x = y;
    y = y';
  }
  return x;
```

(This is not MATLAB code.)

Notice that the iterative (nonfunctional) code, which *assigns* values to variables `y', x` and `y`, must have an additional "workspace", `y'`. The recursive code just uses the parameters to serve as this workspace.

But there is a kind of workspace which functional parameters cannot capture.

This is any variable whose value must be kept *between invocations* of a function.

This is the case for most of the variables in the functions in this week's notes, and we see what a pain MATLAB gives us over them. But all of the examples so far this week need to make the values of these variables available *outside* the functions as well as inside them.

The flipflop program of Week 11 is different. The "state" of the flipflop needs to be known only by the flipflop. The functions that call the flipflop, such as `flipflopRead()` and `flipflopWrite()`, do not need to know the state, say `y`, as we said in Note 1 of Week 11. MATLAB obliged us to write

```
function y = flipflopWrite(data,y)
```

when it should only be necessary to write

```
flipflopWrite(data)
```

We can do this better (but not in MATLAB) by maintaining `y` as a *state* which is held over between invocations—but not available to any code outside of the function for which it is intended.

If we had such a capability, here is how we might wrap up the flipflop function we already wrote in Week 11.

```
statefunction flipflopstateWrite(data)
  state y;
  y = flipflopWrite(data,y)
```

```
end flipflopstateWrite
```

(This is not MATLAB code.)

This is incomplete. `y` is not *initialized* for one thing. We could just say `state y = 1;` to do it.

More importantly, the state cannot be *shared* with another function, say `flipflopstateRead()`.

So we need to *encapsulate* the state and the functions we need to work with it.

```
flipflop
  state y = 1;
  function flipflopstateWrite(data);
  % define the code here (as above)
  function data = flipflopstateRead();
  % define the code here (similarly)
end flipflop
```

(This is not MATLAB code.)

So we could now invoke the flipflop functions in this way:

```
flipflopstateWrite(1);
..  = flipflopstateRead();  % gets 1
..  = flipflopstateRead();  % gets 1
:                           :
flipflopstateWrite(0);
..  = flipflopstateRead();  % gets 0
:                           :
```

8. What if we wanted *two* flipflops?

We'd have to make a *copy* of the state.

(We don't have to copy the *functions*: they are just code. This is for the same reason that we didn't have to copy the `nand()` function when we built the flipflop in the first place. We just used it, repeatedly.)

We can take our above definition of `flipflop` as a template for a *class* of flipflops. If we provide some new syntax, the programmer can *instantiate* as many flipflops as hey likes.

```
f1 = new flipflop;
f1.flipflopstateWrite(1);
..  = f1.flipflopstateRead();  % gets 1
:                              :
f2.flipflopstateWrite(0);
..  = f1.flipflopstateRead();  % gets 1
..  = f2.flipflopstateRead();  % gets 0
:                              :
```

Classes and their instantiation are the heart of (the very badly named) "object-oriented" programming. In fact, although you will hear many opinions about "O-O" programming, only instantiation fundamentally matters.

This kind of programming violates the goal of functional programming, which is to dispense altogether with state, or any hidden "side effects" caused by assignment and update operations.

But it is an improvement over unrestricted states and side effects. It *contains* each state within the module that also contains all the allowed ways of operating on that state. This is called *encapsulation*.

MATLAB can't do any of the coding we've just seen. Can we cobble together a state-preserving,

instantiable flipflop in MATLAB?

Instead of `flipflop`, for which we have already written a whole lot of code which we'd have to modify, let's try the most basic possible class, `counter`.

```
counter
  state ctr = 0;

  function reset
       ctr = 0;
  end reset

  function count
        ctr = ctr + 1;
        return ctr;
  end count
end counter

sheep = new counter;
goats = new counter;
.. = sheep.count;
.. = sheep.count;
.. = goats.count;
sheep.reset;
:
```

We can do this in MATLAB but it will be much longer: we must manage the names and the states ourselves. We can do this using arrays to remember the various instances of the state.

9. Summary

(These notes show the trees. Try to see the forest!)

- Recursion
  - Grammar: processing and evaluating expressions.
  - Fractals: self-similar curves with exponential complexity.
  - Mathematical induction.
  - L-systems: self-similar branching structures.
- Instantiation
  - Workspace vs. state.
  - Functional vs object-oriented programming:
    - functional programming has no side-effects (e.g., assignments), and hence no *state*;
    - object-oriented programming has state, and *instantiation* to copy the state, but it *encapsulates* the state to make it safer;
    - conventional programming (usually called "imperative") allows assignments and state anywhere, thus forcing the programmer to remember all the assigned variables and their values at any given point in the program.

II. The Excursions

You've seen lots of ideas. Now *do* something with them!

1. Combine parsexp with the shuntingYard algorithm and the booleanRevPolEval function of Week 11 to parse and evaluate expressions such as ab+cd.

2. How must the grammar implemented by parsexp be changed to accommodate parentheses in the *input*, such as in a(b+c)d?

3. How many function calls are generated by a function which calls itself thrice and recurses to depth n (the initial call being at level 1, the next three at level 2, and so on to level n).

4. The inventor of chess is legendarily said to have been invited by his grateful head of state to name any reward he wished [Gam47, pp.7–8]. He asked for one grain of wheat for the first square of the chessboard, two for the second, double that for the third, double it again for the fourth square, and so on. How many bushels of wheat did the monarch have to produce?

5. Use MATLAB to calculate the interest growth formula, $v = v_0(1+i)^p$, for the value, v, of a quantity which has grown at interest $i \times 100\%$ per period over p periods. For small interest rates of 1%, 2%, 5%, .., compare this with e^{ip}. Given that $e^{0.72} \simeq 2$, what is a quick way to find out how many years it takes for your money to double if invested at small interest rates?

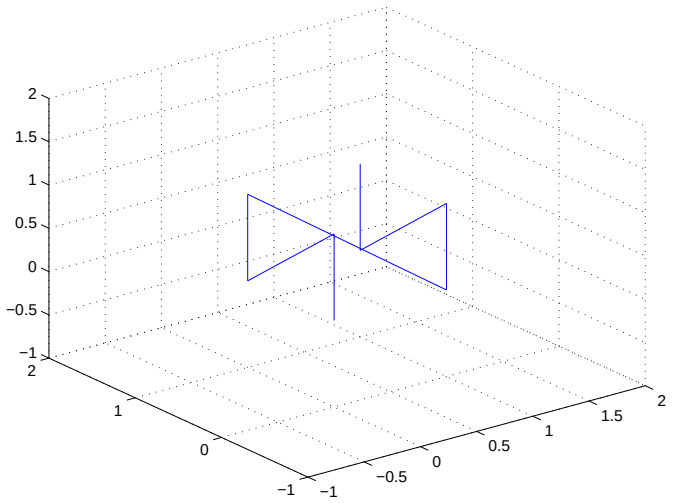
6. The following table gives the gross domestic product (GDP) per capita in 1820 for various regions of the world, and the annual percentage growth rate for each of these regions. (I have interpolated this data from [Sac05, Chap. 2], who cites his sources.)

Region	Western Europe	Eastern Europe	Former USSR	US, Canada Oceania	Latin America	Japan	Asia (not Japan)	Africa
1820 GDP/cap.	1500	800	800	1200	800	800	800	800
% growth/ann.	1.5	1.2	1.0	1.7	1.2	1.9	0.9	0.7

   Use MATLAB to calculate the GDP per capita in 2000, 180 years later, and discuss the poverty gaps.

7. Rewrite peanoStep with a stopping condition for $n == 1$. Why is this not as good?

8. Write a three-dimensional Peano curve drawing program. Here is the basic unit.



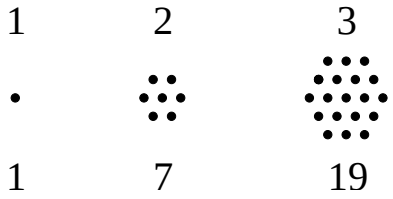
9. Write a two-dimensional Z-order program to draw the Peano curve not using recursion but by interleaving bits of the coordinates.

10. Look up the "Hilbert curve" and write a MATLAB program to draw it to any depth of self-similarity.

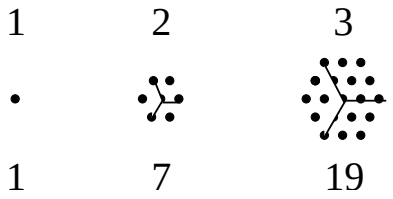
11. a) Find the induction step in a proof that integers that are a multiple of 9 have digits (in the representation based on 10) whose sum is also a multiple of 9.
b) Reverse the induction step from (a) to get a proof of the converse, namely that if the digits of the integer sum to a multiple of 9 then the integer is itself a multiple of 9.
c) Show that, if somebody mistakenly swaps two digits when writing down an integer, the difference between the wrong integer and the correct one is a multiple of 9.
d) Repeat (a) and (b) for multiples of 3 (both the integer and the sum of its digits). Can one of these arguments (i.e., 3 or 9) be deduced from the other?

12. Confirm that Euler's formula, $E + F = V + 2$, is true for a triangle drawn on a sphere, where E is the number of edges (3), F is the number of faces (2: the regions inside and outside the triangle) and V is the number of vertices (3). (For a triangle drawn on a plane, the outside face goes to infinity, so the formula is also true.)
Find the induction steps that show that Euler's formula holds for any polygon or for any collection of polygons.

13. *Hexagonal numbers*, 1, 7, 19, 37, 61, 91, 127, .., are the numbers of dots that can be used to show filled hexagons of sides 1, 2, 3, ..



a) Show by mathematical induction that the sums of hexagonal numbers up to any point, starting at 1, are always perfect cubes (1, 8, 27, 64, ..).
b) Show the same, but this time by visualizing each hexagon as the front three faces of a cube, whose interior can be seen to be filled by all the preceding hexagons.



c) Penrose [Pen94, pp.66–77] argues that proofs such as the above visualization cannot be captured by computational rules such as induction—a visualization can always be found which transcends any currently complete computational system—and hence that mind cannot be programmed. (See the Excursion in Week 11 on non-stopping computations.) Don't take my word for it: read the whole book!
Penrose's argument could explain AI's apparent lack of significant progress, over half a century, towards its main goal, the construction of an intelligent program. AI has, however, provided significant benefits to computer science, in the LISP and Prolog programming languages, the idea of expert systems, techniques used in data mining, and many other paradigms. Penrose does believe that an intelligent machine can be constructed—it just cannot be a program based on current computers.

14. The excursion in Week i about the twenty-five primes less than 100, and the follow-up excursion in Week iii, give two tricks for finding whether integers are multiples of certain other integers, depending on the base b in which the calculations are done. First, the base-b "digits" of any integer which is a multiple of $b+1$ alternate-sum to zero (e.g., multiples of 11 in base 10), and conversely. Second, the base-b "digits" of any integer which is a multiple of $b-1$, or of its divisor, sum to a multiple of $b-1$, or of the same divisor (e.g. 3 and 9 in base 10), and conversely.
Use induction to prove these rules both ways. E.g., show that if an integer is a multiple of 11 (base 10) then its digits alternate-sum to 0, and also that if the digits of an integer (base 10) alternate-sum to zero then the integer is a multiple of 11.

15. Use induction to show that the difference $f_n^2 - f_{n-1}f_{n+1}$ of terms in the Fibonacci sequence (Week ii) alternates between $+1$ and -1.

16. a) Given that the area of a triangle with vertices $(x_1, y_1), (x_2, y_2)$ and (x_3, y_3) is

$$\frac{1}{2}\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix} = (x_1y_2 - x_2y_1) + (x_2y_3 - x_3y_2) + (x_3y_1 - x_1y_3) = \sum_{j=1}^{3} x_jy_{j+1} - x_{j+1}y_j$$

(Week 8 Note 6) where, in the last sum, $j+1 = 4$ wraps around to $j+1 = 1$, use induction to show that the area of any polygon with vertices $(x_1, y_1), .., (x_n, y_n)$ is

$$\frac{1}{2}\sum_{j=1}^{n} x_jy_{j+1} - x_{j+1}y_j$$

where $j+1 = n+1$ wraps around to $j+1 = 1$.
Hint. The sum must follow the vertices of the polygon in counterclockwise order. Change a polygon of $k-1$ sides to one of k sides by drawing a triangle based on one of the sides and adding the triangle area to the area of the original polygon: what gets cancelled?
b) A similar argument can be used to find the *centroid* of any polygon, which is the point on which the polygon would balance if it were made by cutting it out of a sheet of material with uniform density. The x coordinate of the centroid is the average value of x taken over the whole polygon, and the y coordinate is the average value of y.
Suppose that the centroid of a triangle is (see (f) in the next excursion)

$$\frac{1}{6A}\left(\sum_{j=1}^{3}(x_j + x_{j+1})(x_jy_{j+1} - x_{j+1}y_j), \sum_{j=1}^{3}(y_j + y_{j+1})(x_jy_{j+1} - x_{j+1}y_j)\right)$$

where A is the area of the triangle as in (a) above. Use induction to show that the centroid of any polygon is

$$\frac{1}{6A}\left(\sum_{j=1}^{n}(x_j + x_{j+1})(x_jy_{j+1} - x_{j+1}y_j), \sum_{j=1}^{n}(y_j + y_{j+1})(x_jy_{j+1} - x_{j+1}y_j)\right)$$

where A is the area of the polygon as in (a) above.
What difference does it make if j increases in a clockwise direction or if it increases in a counterclockwise direction? Why? How does this relate to considerations leading to the "interval algebra" (Week 7c, Notes 6 and 1)?
c) Convince yourself that, in general,

$$\sum_{j=1}^{n}(\text{any quantity symmetric in } j \text{ and } j+1)(x_jy_{j+1} - x_{j+1}y_j)$$

is true for n if it is true for $n = 3$. This and the special cases in (a) and (b) above illustrate the Kelvin-Stokes theorem. Look this up! (You will need integral calculus and the differential calculus of vectors. The next excursion is a very preliminary introduction to differential and integral calculus.)
The magic in the Kelvin-Stokes theorem is that it enables us to calculate quantities involving the whole of a region (polygon) by looking *only at the boundary.*

17. **Calculus.** a) In Week 8 Note 10 we encountered the *slope*s of functions and found

$$\text{slope } c = 0; \;\; \text{slope } x = 1; \;\; \text{slope } x^2 \simeq 2x$$

More generally, we can use the binomial theorem to show

$$\text{slope } x^k \;\; = \;\; \frac{(x+\Delta x)^k - x^k}{\Delta x} = \frac{(x^k + kx^{k-1}\Delta x + \cdots) - x^k}{\Delta x} \simeq kx^{k-1}$$

We also saw (Excursions of Week 8) that

$$\text{slope } \sin(\theta) = \cos(\theta); \;\; \text{slope } \cos(\theta) = -\sin(\theta)$$

Suppose that k above can be a negative integer and show that

$$\text{slope } \tan(\theta) = \sec^2(\theta)$$

b) We can also find slopes of slopes (slope^2), slopes of slopes of slopes (slope^3) and so on. Thus

$$\text{slope}^2 x^2 \simeq 2; \;\; \text{slope}^2 x^k \simeq k(k-1)x^{k-2}; \;\; \text{slope}^3 x^k \simeq k(k-1)(k-2)x^{k-3}; \;\; \text{slope}^2 \sin(\theta) \simeq -\sin(\theta)$$

What is slope$^4\sin(\theta)$? What is slope$^2 e^{i\theta}$ (see Week 4 Note 9)?
c) Use arguments similar to those in (a) to show that

$$\begin{aligned} \text{slope } f(x) + g(x) &\simeq \text{slope } f(x) + \text{slope } g(x) \\ \text{slope } f(x)g(x) &\simeq g(x)\text{slope } f(x) + f(x)\text{slope } g(x) \\ \text{slope}_x f(g(x)) &\simeq \text{slope}_y f(y)\text{slope}_x g(x) \end{aligned}$$

(Hint for the last, the *chain rule*: let g be $g(x)$ and $g + \Delta g$ be $g(x + \Delta x)$, and explore.)
d) The *antislope* operator maps a function to the function whose slope it is. From (a) we can work out some examples.

$$\text{antislope}_x \; c \simeq cx \;\; \text{antislope } x \simeq x^2/2; \;\; \text{antislope } x^2 \simeq x^3/3$$

But wait! The antislope is not unique. Since the slope of any constant function is 0, we can add an arbitrary constant to the result of the antislope operator. Mathematicians usually call this C or c, but we can call it *zeroslope* or *0slope* or, if we need more than one, $z_1, z_2, ...$

$$\text{antislope}_x \; c \simeq cx + \textit{0slope}; \;\; \text{antislope } x \simeq x^2/2 + \textit{0slope}$$

What is antislope$^2 c_0$?
Show that antislope slope $f(x) = f(x) =$ slope antislope $f(x)$.
e) Antislope and area are almost magically related. This is the Fundamental Theorem of Calculus. First, the change in the area under $f(x)$ as x extends to $x + \Delta x$ is approximately

$$\Delta\text{area} = \frac{f(x+\Delta x) + f(x)}{2}\Delta x \simeq f(x)\Delta x$$

and so we can call the slope of the area

$$\text{slope}_x \text{ area} = \Delta\text{area}/\Delta x \simeq f(x)$$

Compare this with: slope antislope $f(x) = f(x)$.
Second, the area from a to b under slope $f(x)$ (to keep the equation below manageable, we abbreviate slope $f(x)$ as $f'(x)$) can be estimated by looking at equally-spaced successive values of x: $x_0 = a, x_1, x_2, .., x_n = b$

$$\begin{aligned} \text{area slope } f(x) &\simeq f'(x_1)\Delta x + f'(x_2)\Delta x + .. + f'(x_n)\Delta x \\ &= (\frac{f(x_1) - f(x_0)}{\Delta x} + \frac{f(x_2) - f(x_1)}{\Delta x} + .. + \frac{f(x_n) - f(x_{n-1})}{\Delta x})\Delta x \\ &= f(b) - f(a) \end{aligned}$$

(Note the resemblance of the cancellation of all the intermediate points to the cancellation of the polygon edge in the previous excursion when we add a new triangle. Turn this argument into a proof by induction! The magic is similar to the magic of Kelvin-Stokes: a whole area can be found by looking only at the end points. Note that the total area is basically the sum of a whole bunch of small areas.)
Compare this with: antislope slope $f(x) = f(x)$.
The magic here is that the whole area from a to b under slope $f(x)$ depends only on $f(b) - f(a)$. The practical result is that the area from a to b under $f(x)$ is

$$\text{antislope } f(x) \mid_{x=b} - \text{antislope } f(x) \mid_{x=a} .$$

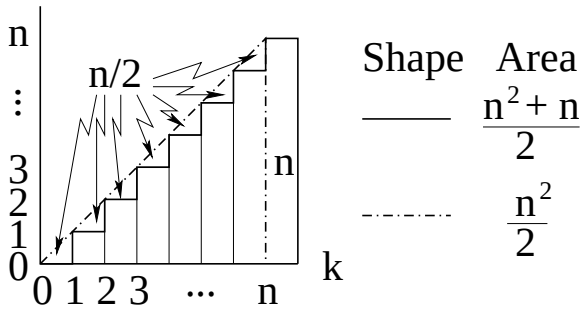
Note that we can forget about the arbitrary constants because we took the difference.
f) For $f(x) = x$ let's relate

$$\text{antislope } x \mid_{x=n} - \text{antislope } x \mid_{x=0} = n^2/2$$

to the sum (Week i, Notes 1,2)

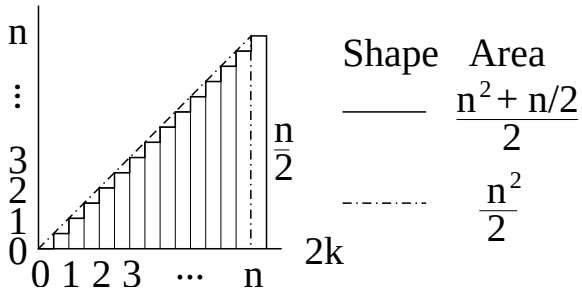
$$\Sigma_{k=0}^{n} k = n(n+1)/2 = (n^2 + n)/2.$$

(why does it make no difference if we start the sum at 0 or at 1?)
If we multiply each term k by the width w separating it from the next term, $w = 1$, we get an approximation to the area under the triangle.



Now let's split each strip into two strips of only half the width, $w = 1/2$. The new area is

$$\frac{1}{2}\Sigma_{k=0}^{2n}\frac{k}{2} = \frac{1}{2}\frac{2n(2n+1)}{4} = \frac{(n^2 + n/2)}{2}.$$



We are closer.
Show that splitting each of the original strips into m strips of width $w = 1/m$ each gives

$$\frac{1}{m}\Sigma_{k=0}^{mn}\frac{k}{m} = \frac{1}{m}\frac{mn(mn+1)}{2m} = \frac{(n^2+n/m)}{2}.$$

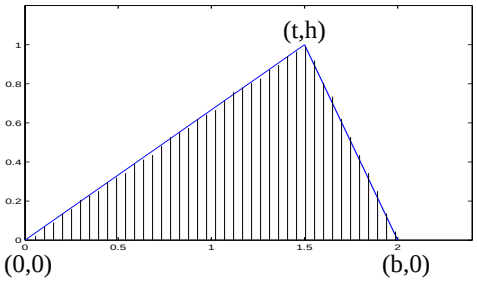
so as m becomes arbitrarily large, the area calculated by summing $f(x)\Delta x$ becomes arbitrarily close to the area given by the antislope.
Check that this is also true for the area under x^2 from 0 to n, using the antislope and the n^2 summation formula from Week i, Notes 1 and 2.
g) As examples, let's find the area under the triangle with vertices $(0,0), (b,0), (t,h)$, and then the average values of x and y in this triangle to find the centroid needed in (b) of the previous excursion.
The equations of the lines from (0,0) to (t,h) and from $(b,0)$ to (t,h) are, respectively,

$$y = hx/t \qquad \text{and} \qquad y = (x-b)h/(t-b)$$

The antislopes are, respectively (omitting the *0slope* constants)

$$hx^2/2t \qquad \text{and} \qquad x(x/2-b)h/(t-b)$$

and these must be evaluated at $x = t$ and $x = 0$ for the first and at $x = b$ and $x = t$ for the second.
Show that the differences sum to $bh/2$, i.e., the usual triangle area of half base times height.



(Note that we can choose the x- and y-axes so that the triangle has these particular vertices without loss of generality. But the above calculation assumes $0 \le t \le b$: re-do it for the other two possibilities, $t < 0$ or $b < t$.)
The second example is to find the centroid. For the x-coordinate we must take the antislopes of, respectively

$$y = hx^2/t \qquad \text{and} \qquad y = x(x-b)h/(t-b)$$

and evaluate these as above. Show that this gives

$$(b+t)bh/6$$

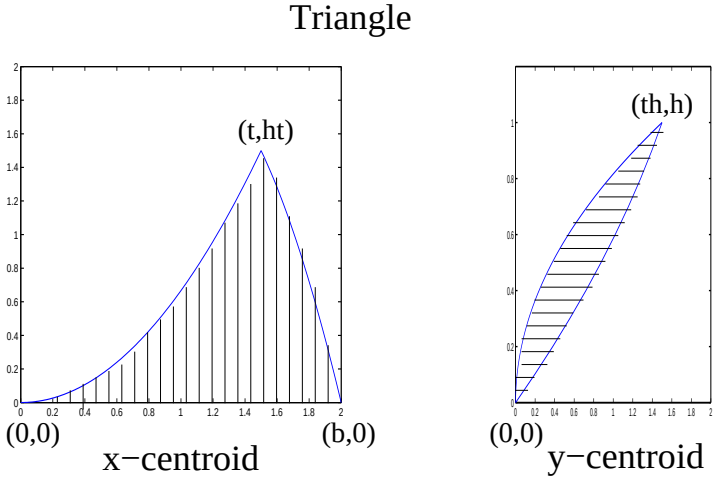
which when divided by the area gives the x-coordinate of the centroid. Check that this makes sense for $t = b/2$.
For the y-coordinate we must re-write the line equations in terms of y

$$x = ty/h \qquad \text{and} \qquad x = (t-b)y/h + b$$

respectively, and then find the antislope of the difference

$$(t-b)y^2/h + by - ty^2/h$$

and evaluate this at $y = h$ and $y = 0$.
Finally show that these calculations give the same centroid coordinates as the sums in (b) of the previous excursion, applied to the triangle $(0,0), (b,0), (t,h)$.



h) As a further example, let's confirm that the area of a circle is πR^2. We start by measuring angles in units based on the length of the part of the circumference of the circle of radius R swept out by angle θ. Since π is defined as the ratio of full circumference to diameter of any circle, this length is $2\pi R$ for a full rotation, πR for an about-face, and $\pi R/2$ for a right-angle. It makes sense to define a unit angle (called a *radian*) such that the arc of circumference of a unit circle ($R = 1$) traced by that angle has length 1. In these units we can say that

$$r\Delta\theta\Delta r$$

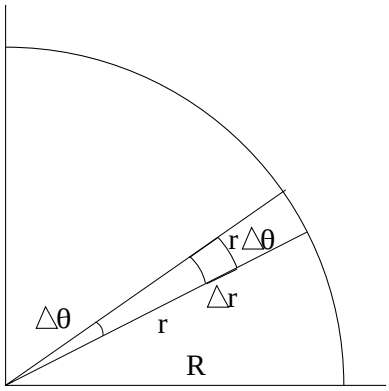
is the small area between two radii separated by angle $\Delta\theta$ and between distances r and $r+\Delta r$ from the centre. To sum all these little areas up for a quarter-circle of radius R, we need to evaluate

$$\text{antislope}_r r\Delta\theta\Delta r = \frac{1}{2}r^2\Delta\theta$$

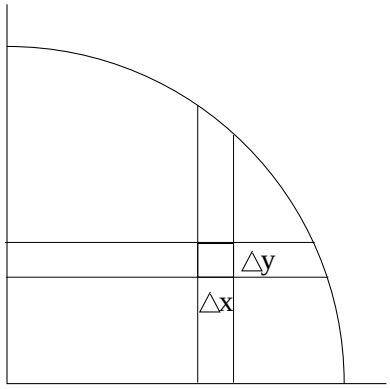
at 0 and R, and evaluate the θ-antislope of that

$$\text{antislope}_\theta \frac{1}{2}R^2\Delta\theta$$

at 0 and $\pi/2$, getting $\pi R^2/4$ for the quarter-circle, and so πR^2 for the full circle.



A similar two-dimensional antislope calculation can be done on $\Delta x \Delta y$ for $0 \leq y \leq \sqrt{R^2 - x^2}$ and then $0 \leq x \leq R$ for the same quarter-circle, but we haven't discussed finding antislopes of square roots. We should get the same answer, $\pi R^2/4$, and this could tell us how to calculate such antislopes: geometrical arguments and alternative approaches are often the basis of "tables of integrals" that can be looked up for antislope calculations.



i) You will need conventional terminology and notation if you explore calculus further, although "slope" and "antislope" should be more suggestive and clear for beginners.

slope	derivative	$\frac{\mathrm{d}}{\mathrm{d}x}$
slope_x, slope_y	partial derivative	$\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$
antislope	integral	$\int \mathrm{d}x$
area	definite integral	$\int_a^b \mathrm{d}x$

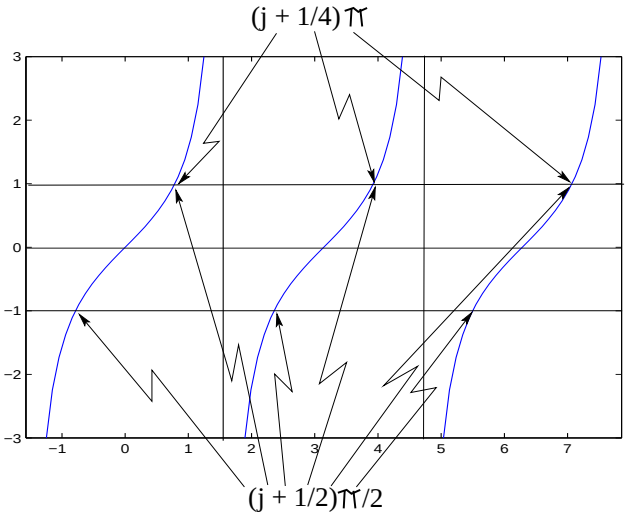
In this notation, the statement of the Kelvin-Stokes theorem of the previous excursion is, for any continuous and differentiable functions $P(x, y)$ and $Q(x, y)$,

$$\int\int (\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}) \mathrm{d}x \mathrm{d}y = \oint (P \mathrm{d}x + Q \mathrm{d}y)$$

where the double integral is definite, over a given two-dimensional region and the integral with a circle on it is also definite, over the closed boundary of the given region. You will need practice and more study to understand this equation.

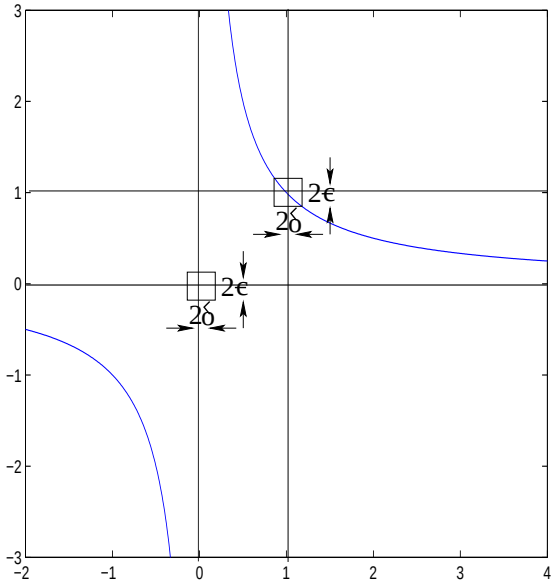
j) Note that the slope operator gives linear approximations to functions, whether linear or not themselves, at any point to which it applies. Differential calculus thus connects the nonlinear math of Weeks 10–12 with the linear math of Weeks 1–9.

18. **Continuity.** The "magic" of the last two excursions seems to be very powerful. In each case, quantities involving the interiors of regions can be calculated by looking only at the boundaries. This magic must come from very strong assumptions, and it is only fair to bring these to light.
The strong assumption in each case is that of *continuity*.
a) We start with a function where the sum over all the equally-spaced values ((e) of the previous excursion) does *not* give the area under the function. Consider $\tan(x)$ evaluated at $x_j = (j + \frac{1}{4})\pi$ (which is 1) for $j = 0, .., n$. Plot $\tan(x)$ from 0 to, say, 2.5π and see that the true area under the function is not $n \times 1$. Refine the calculation by using $x_j = (j + \frac{1}{2})\frac{\pi}{2}$: the values alternate from 1 to -1. But the area under the curve is not 0.



b) In fact, $\tan(x)$ is *discontinuous* at $x = (j + \frac{1}{2})\pi$ for all j. We must refine all our above definitions of slope and antislope by taking *limits* as Δx becomes arbitrarily small. Then the $\simeq$ becomes $=$ in all the equations. For its slope to exist everywhere, a function must at least be continuous everywhere. How can we check this?
A check for continuity is the "epsilon-delta" test, which is best seen as a challenge-game for two players. Player D, who is claiming continuity at point (a, b), tells player E, who is challenging: "I will concede that function f (i.e., $y = f(x)$) is not continuous at (a, b) $(b = f(a))$ if you can find a positive number ϵ with $\mid y - b \mid < \epsilon$ such that I cannot find a positive number δ which matches it by making $\mid x - a \mid < \delta$."
Let's try this with $f(x) = 1/x$ at point (1,1) on the function curve.
E: "I choose ϵ to be 0.1"
D: "OK, I match with $\delta = 0.09$"
D followed a procedure to get this δ: E is allowing y to be in the range $1 - \epsilon < y < 1 + \epsilon$, i.e., $0.9 < y < 1.1$. This puts x in the range $1/1.1 < x < 1/0.9$, i.e., $0.9090.. < x < 1.1111...$ So if $\delta = 0.09$, as D said, the range of x is $1 - 0.09 < x < 1 + 0.09$, i.e., $0.91 < x < 1.09$. This makes the range of y to be $1/1.09 < y < 1/0.91$ which is $0.9174.. < y < 1.0989..$ and this is tighter than the range E originally specified, and y will be in this range for any x in the range D specified, so D wins.
E can now say "I choose ϵ to be 0.01", but D can follow the same procedure and find a winning δ. D can do this for any ϵ E choses, so E cannot win and D need not concede that $1/x$ is not continuous at (1,1). Hence, the epsilon-delta test says $1/x$ is continuous at (1,1).
Now let's try the game with $f(x) = 1/x$ at $x = 0$. Woops, $1/x$ is not defined for $x = 0$, so we'll have to complete the function, say by including point (0,0) on it. So we'll test continuity at point (0,0).
E: "I choose ϵ to be 0.1"

Now D is stuck, because the procedure will give $-10 < x < 10$ so D could choose $\delta = 11$. This will satisfy ϵ for the extremes of $y = 1/x$ namely $y = -1/10 = -0.1$ and $y = 1/12 = 0.0833..$ but it will not satisfy ϵ for x in the middle of its range, say $x = 1$: y will then be 1, which is bigger than ϵ.
The whole epsilon-delta procedure amounts to trying to contain $f(x)$ in a box centered at (a, b) with height 2ϵ, chosen by E, and width 2δ, sought by D to match the ϵ and contain all intermediate values of the function.



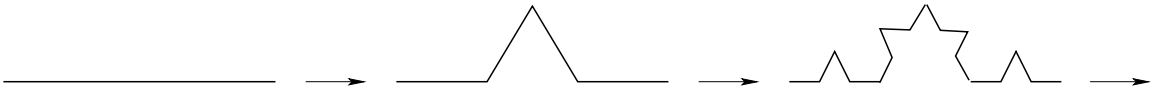
Find the procedure for D to defend continuity of $\tan(x)$ everywhere except at $x = (i + \frac{1}{2})\pi$ for any i, and use it to show that $\tan(x)$ is not continuous at these points.
Use epsilon-delta to show that the *step function*

$$f(x) = \lfloor x \rfloor = \text{the integer value that} = x \text{ or is next below} x$$

is discontinuous at integer values of x but otherwise continuous.
Show that the slope of an absolute value,

$$slope \mid x \mid$$

is discontinuous at (0,0) but otherwise continuous.
c) Fractals can exhibit discontinuity not only at a single point or at a mere countable infinity of discrete points, but everywhere. The *triadic Koch curve*, while continuous everywhere, has no slope anywhere, or, we could say, has slope which is discontinuous everywhere.
The initiator for the triadic Koch cuve is a straight line segment, and the generator step divided this segment into three and replaces the middle third by two edges of the same length, producing a four-piece segment which is no longer straight. Each of the four pieces is in turn subdivided into three and each subdivision is transformed in the same way. This process is repeated ad infinitum resulting in the continuous but slopeless curve.



d) The idea of continuity is one of the most imaginative inventions of mathematics, and the epsilon-delta test captures continuity remarkably. Look into the history of the idea of continuity.
Continuity requires us to include irrational numbers, which have infinitely long decimal representations and so, if nature abhors infinities, continuity is unlikely to occur in nature. But continuity is a very powerful abstraction as we began to see in the previous excursion. Where it can be applied to model physical situations, continuous mathematics supports important deductions and is a useful complement to the discrete math and programming we have been using in these excursions so far.
Note that "algebraic" irrationals such as the square root of two can be finitely specified simply by stating the algebraic equation that they satisfy.
Give a finite specification of π, which is a "transcendental" irrational satisfying no algebraic equation.
Metaphysics is the philosophy of disputing what is truly "real". Plato said that mathematical abstractions really exist in some intellectual space: there is, for instance, an ideal circle somewhere, according to platonists, from which everybody's idea of a circle is derived. Find out if most mathematicians are platonists and, if so, suggest why!
(We do not dwell on metaphysics in these excursions, but must confess that metaphysics is involved right from their beginning, namely the Popperian epistemology in Week 1. Perhaps you would best wait until you are sixty before indulging in this branch of philosophy.)

19. **Series.** In part (g) of the excursion on calculus, above, we encountered $\sqrt{R^2 - x^2}$ and did not try to find the area under it. If we could do so, we would have a way of calculating π, by comparing the result with the area we found using polar coordinates.
Let's work on the related function, $\sqrt{1+x}$.
(How can you express $\sqrt{R^2 - z^2}$ in terms of $\sqrt{1+x}$?)
a) Since this can be written $(1+x)^{1/2}$, let's think about $(1+x)^m$ more generally. For m an integer, this leads to the binomial coefficients. Here they are (ignore the Aha! and Oho! columns for the moment).

m	Oho! -1	0	1	2	3	4	..	m	Aha! 1/2
x^0	1	1	1	1	1	1		1	1
x^1	-1		1	2	3	4		"$\times m/1$	$-1/2$
x^2	1			1	3	6		"$\times(m-1)/2$	$1/8$
x^3	-1				1	4		"$\times(m-2)/3$	$-1/16$
x^4	1					1		"$\times(m-3)/4$	$5/128$
:	:							:	:

Work out $(1+x)^0, (1+x)^1, (1+x)^2, (1+x)^3$ and $(1+x)^4$ to confirm these coefficients. While doing this, find the induction step for the proof that the coefficient of x^n in the expansion of $(1+x)^m$ is the sum of two previous coefficients (of x^{n-1} and x^n in $(1+x)^{m-1}$). (Note that this is a general result if we suppose all coefficients not shown to be zero, i.e., those for $n < 0$ and $n > m$.)
b) The Aha! column is Isaac Newton's aha! [Wes80, p.117]: he wondered about using the same formula for non-integer m. If this works, we have an expansion for $(1+x)^{1/2} = \sqrt{1+x}$

$$\sqrt{1+x} = 1 - \frac{x}{2} + \frac{x^2}{8} - \frac{x^3}{16} + \frac{5x^4}{128} - ..$$

The big jump is that this expansion is infinite: there is a term for any non-negative power of x. Such a sum is called a *series* and this one is infinite. Newton did not blanch.
Show that the induction step you found in (a) is true even for non-integer m.
c) Even a Newton aha! is not necessarily a proof. We should at least persuade ourselves that

this infinite series does give $\sqrt{1+x}$. We can do this by squaring the series.
Maybe we should square any infinite series while we are at it. That will also help with the series for $(1+x)^{3/2}$ ($m = 3/2$) and other half-integers. Let the series be $1+a_1x+a_2x^2+a_3x^3+..$ and its square be $1 + A_1x + A_2x^2 + A_3x^3 + ..$

	1	a_1	a_2	a_3	a_4	a_5	..
1	1	a_1	a_2	a_3	a_4	a_5	
a_1	a_1	a_1^2	a_1a_2	a_1a_3	a_1a_4	a_1a_5	
a_2	a_2	a_1a_2	a_2^2	a_2a_3	a_2a_4	a_2a_5	
a_3	a_3	a_1a_3	a_2a_3	a_3^2	a_3a_4	a_3a_5	
a_4	a_4	a_1a_4	a_2a_4	a_3a_4	a_4^2	a_4a_5	
a_5	a_5	a_1a_5	a_2a_5	a_3a_5	a_4a_5	a_5^2	
:							

The coefficients in the square are just the diagonal sums of these entries, starting at 1 then moving down-and-right in a wave, $A_1 = a_1 + a_1$, $A_2 = a_2 + a_1^2 + a_2$ and so on.
Check this, and then use it to find the coefficients for $(1 + x)^{1/2}$: $A_1 = 1$, $A_2 = 0, A_3 = 0, ...$
Show that these agree with Newton's aha!
Show that the first term in the expansion of $(1 + x)^{1/k}$ must be $1/k$.
c) We now have expansions for $(1 + x)^{\ell/k}$ for any non-negative rational ℓ/k. What about negative powers? $1/(1 + x) = (1 + x)^{-1}$ brings us to the Oho! column, a second from Newton [Wes80, p.120]. This time, Newton went directly to the theorem you proved in (a) and said that the coefficient of x^n in the expansion of $(1+x)^0$ must be the sum of two previous coefficients (of x^{n-1} and x^n in $(1 + x)^{-1}$).
Considering all coefficients of $(1 + x)^0$ to be zero, except for the single 1, show that the Oho! column follows from the $m = 0$ column and this rule.
Multiply the resulting series by $1/(1 + x)$ to check.
d) Now that Newton had ways to represent $(1+x)^{\ell/k}$ for any rational ℓ/k, he could differentiate and integrate such expressions. Show that the expansions for the slope is given by the expansions in the table in (a) above, but with the x^n in the first column replaced by nx^{n-1}.
Show that the expansions for the antislope change x^n to $x^{n+1}/(n + 1)$.
e) Use these results to find the antislope of $\sqrt{1 - x^2}$ and hence to find the area under this curve from 0 to 1. Use this to calculate π to as many places as you wish. Turn it into a MATLAB program to calculate it further.
f) Note that there is no slope that equals $1/x$. So what is the antislope of $1/x$? Try finding the antislope of $1/y$ where $y = 1 + x$.
g) Functions that can be represented as series, finite or not, of powers of x, can be calculated just as you calculated π in (e). Suppose $f^{(1)}(x)$ is the slope of $f(x)$, $f^{(2)}(x)$ is the slope of the slope, $f^{(3)}(x)$ is the slope of that, and so on. (Recall that the existence of each of these slopes requires the continuity of the function it is the slope of, and that this is a strong assumption.) Then , if

$$f(x) = a_0 + a_1x^1 + a_2x^2 + a_3x^3 + ..,$$

show that $a_0 = f(0)$, $a_1 = f^{(1)}(0)$, $a_2 = f^{(2)}(0)/2$, .. $a_k = f^{(k)}(0)/k!$, and so that for small x,

$$f(x) = a_0 + f^{(1)}(0)x + f^{(2)}(0)x^2/2 + f^{(3)}(0)x^3/6 + ..$$

Use this "Taylor's series" to find expansions for $\sin(x)$, $\cos(x)$, e^{ix} where $i^2 = -1$, and e^x.
Use a method such as that of (b) above to find the expansion for the inverse of a function for which an expansion is known. Use this to find the expansion for the inverse of e^x, which is $\ln(x)$ the "natural logarithm" of x. Compare your result with your result for (f) above, recalling that (f) dealt with a function of $1 + x$.
h) Getting into infinite series requires us to think about *convergence*. Does the series

$1 - x + x^2 - x^3 + ..$ have a finite value for *any* value of x in (c) above? Try some values for x to explore this question. What about the other series in this excursion, especially the series for π? What about the series $1/x : 1/1 + 1/2 + 1/3 + 1/4 + ..$?

20. Our program for gcd is not totally safe. It needs to be protected against $x < y$ and against negative inputs. Using only if statements and further calls to gcd itself, add these protections to the code.

21. Express factorial as a recursive function. What are the starting and inductive steps needed to prove that your approach is correct?

22. Look up the legend of the "Towers of Hanoi" and write a recursive program to end the Universe.

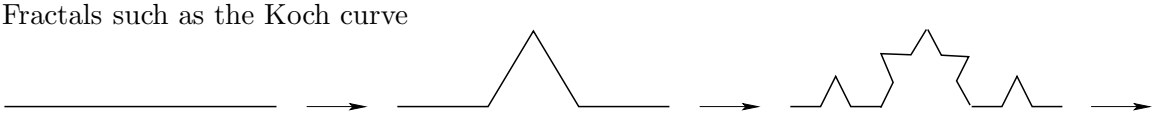
23. Look up John McCarthy (1927–) and the functional programming language LISP. How can the syntax of a language be identical to the syntax of the data it processes? Write a LISP function to reverse the elements of a list.

24. Rewrite the branch-drawing program of Note 6 using `gplot()` in MATLAB.

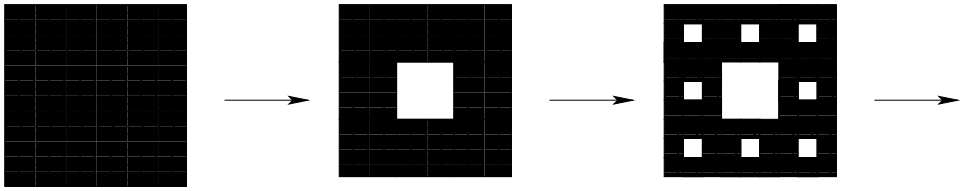
25. Look up Prusinkiewicz and Lindenmayer's *The Algorithmic Beauty of Plants* [PL90] and write MATLAB programs for some of the other branching structures on p.25.

26. Write a translator to convert the symbolic strings, that give rewriting rules, into programs to draw the recursive pictures. (Advanced skills and a better language than MATLAB are required.)

27. We first encountered the idea of dimension in this course as the number of orthogonal (or, more generally, linearly independent) vectors in the basis of a space, i.e., the number of coordinates needed to locate a point in the space. The dimension of a space, from this point of view, is thus an integer.
Fractals such as the Koch curve



or the *Sierpinski carpet*



introduce non-integer dimensions. Let's ask what happens as the scale of measurement changes, for ordinary figures such as edge, square or cube, and for these two fractals. We generalize the terms "length", "area", "volume" and so on to, simply, "measure", which we give in the table below for a variety of *scale*s, s, which depend on the *resolution*, r. The measure of edge and Koch curve will be their length, of square and Sierpinski carpet will be their area, and of cube will be its volume. In the examples in this table, the linear scale is refined by a factor 1/3 at each step. We start with scale $s = 1$ at the lowest resolution, $r = 0$, then $s = 1/3$ at the next resolution, $r = 1$, and so on.

r	s	edge	square	cube	Koch	Sierpinski
0	1	s	s^2	s^3	s	s^2
1	1/3	$3s$	3^2s^2	3^3s^3	$4s$	$8s^2$
2	1/9	3^2s	$(3^2)^2s^2$	$(3^3)^2s^3$	4^2s	8^2s^2
:	:	:	:	:	:	:
r	$1/3^r$	3^rs	$(3^2)^rs^2$	$(3^3)^rs^3$	4^rs	8^rs^2

Note that for the non-fractals, edge, square, cube, etc., the general form is $(3^d)^r s^d$ where d is the number of dimensions, 1, 2, 3, etc. For the Koch curve, the form is $4^r s$, which we could call $(3^d)^r s$ if we say $3^d = 4$, i.e., $d = \log 4/\log 3 = 1.2618595$. For the Sierpinski carpet, the form is $8^r s^2$, which we could call $(3^d)^r s^2$ if we say $3^d = 8$, i.e., $d = \log 8/\log 3 = 1.8927893$. (Note the subtlety that we did not use the same d as the power of s in these calculations: the power of s is the *topological* dimension, and this is 1 for the Koch curve, which can be straightened into a line, and 2 for the Sierpinski carpet, which can be squeezed into a plane.)
Look up the Sierpinski triangle and calculate its dimensionality.
What is the (*fractal*) dimension of the Peano curve?
Construct a fractal of dimension 1.5.
In the limit, what is the length of the Koch curve? the area of the Sierpinski carpet?
Find a fractal of dimension < 1. What is its measure? Where is it discontinuous?

28. Write a set of MATLAB functions to mimic the object-oriented Counter class without using arrays as storage. Explain why not using the arrays is a little misleading in a general discussion of the advantages of automatic instantiation.

29. Any part of the lecture that needs working through.

References

[Gam47] George Gamow. *One two three ... infinity: Facts & Speculations of Science*. The Viking Press, New York, 1947.

[Pea90] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Math. Ann.*, 36:157–60, 1890.

[Pen94] Roger Penrose. *Shadows of the Mind*. Vintage (Random House), London, 1994.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990. Electronic version 2004: algorithmicbotany.org/papers/#abop.

[Sac05] Jeffrey D Sachs. *The End of Poverty: Economic Possibilities for Our Time*. Penguin Group (USA) , Inc, New York, 2005.

[Wes80] Richard S Westfall. *Never at Rest: A Biography of Isaac Newton*. Cambridge University Press, New York, 1980.