
Programming RuFaS

— A Tour of the Coding Process —

Pseudocode

f _x = 0.00001*(HB16*(1/HL56-1)-HE16)											
C	D	GR	GS	GT	GU	GV	GW	GX	GY	GZ	HA
								Layer 1	Layer 2	Layer 3	
		Initial residue	0				Labile P	31.5	31.5	31.5	
		Fresh N mineral rate	0.05				Fresh P	0	0	0	
							Fraction active	0.02	0.02	0.02	
							active min. rt	0.0003	0.0003	0.0003	

Mineralization											
year	yday	ue	kg/ha residue	nd.c.n	nd.c.p	nd.factor	decay rate	active N mineralization (kg/ha)	Fresh min Layer 1	Fresh Decomp Layer 1	
1	244	0.0	0	0	0	1.000	0.014392627	0.00310881	0.00252728	0.00202887	0
1	245	0.0	0.0	0	0	1.000	0.013415775	0.00289756	0.00246266	0.00205356	0
1	246	0.0	0.0	0	0	1.000	0.012833325	0.00277154	0.00245243	0.00205644	0
1	247	0.0	0.0	0	0	1.000	0.012706385	0.00274391	0.00250595	0.002105	0
1	248	0.0	0.0	0	0	1.000	0.012494843	0.00279627	0.00262554	0.00220262	0
1	249	0.0	0.0	0	0	1.000	0.013020975	0.00281327	0.00271541	0.00227565	0
1	250	0.0	0.0	0	0	1.000	0.013633895	0.00294482	0.00290388	0.0024222	0
1	251	0.0	0.0	0	0	1.000	0.013041716	0.00281544	0.00285487	0.00239214	0
1	252	0.0	0.0	0	0	1.000	0.013113068	0.00283409	0.00293952	0.00246071	0
1	253	0.0	0.0	0	0	1.000	0.01314665	0.00288082	0.00304628	0.0025454	0
1	254	0.0	0.0								
1	255	0.0	0.0								
1	256	0.0	0.0								
1	257	0.0	0.0								
1	258	0.0	0.0								
1	259	0.0	0.0								
1	260	0.0	0.0								
1	261	0.0	0.0								

```

pminrl.f
80
81 IF (PBAL(I) .GE. 0.0) THEN
82
83 PFLOWR(I) = 0.0
84 CNTDAY(I) = 0.0
85
86 IF (PBAL(I) .GT. OLDPBAL(I))
87 IF (DAYS(I) .GE. 1.0) DAY
88 IF (DAYS(I) .EQ. 0.0) DAY
89
90 PSPFAC(I) = VARA(I) * (DAYS(I) ** VARB(I))
91
92 PFLOW(I) = pspfac(I) * PBAL(I)
93
94 IF (PFLOW(I) .GT. LABILEP(I)) PFLOW(I) = LABILEP(I)
95
96 LABILEP(I) = LABILEP(I) - PFLOW(I)
97 IF (LABILEP(I) .LE. 0.0) LABILEP(I) = 0.0
98
99 ACTIVEP(I) = ACTIVEP(I) + PFLOW(I)
100 IF (ACTIVEP(I) .LE. 0.0) ACTIVEP(I) = 0.0
101
102 OLDPBAL(I) = PBAL(I)
103
104
105 ENDF

```

$$a_{C:P} = \frac{0.58 \cdot r_{sd,i}}{orgP_{soil,i} + P_{minsoil,i}} \quad 3:1.2.6$$

where $a_{C:P}$ is the C:P ratio of the residue in the soil layer, $r_{sd,i}$ is the residue in layer ly (kg/ha), 0.58 is the fraction of residue that is carbon, $orgP_{soil,i}$ is the phosphorus in the fresh organic pool in layer ly (kg P/ha), and $P_{minsoil,i}$ is the amount of phosphorus in solution in layer ly (kg P/ha).

The decay rate constant defines the fraction of residue that is decomposed. The decay rate constant is calculated:

$$\delta_{w,i,ly} = \beta_{w,i} \cdot \gamma_{w,i,ly} \cdot (\gamma_{w,i,ly} \cdot \gamma_{w,i,ly})^2 \quad 3:1.2.7$$

where $\delta_{w,i,ly}$ is the residue decay rate constant, $\beta_{w,i}$ is the rate coefficient for mineralization of the residue fresh organic nutrients, $\gamma_{w,i,ly}$ is the nutrient cycling residue composition factor for layer ly , $\gamma_{w,i,ly}$ is the nutrient cycling water factor for layer ly , and $\gamma_{w,i,ly}$ is the nutrient cycling water factor for layer ly .

The nutrient cycling residue composition factor is calculated:

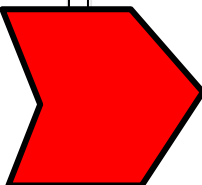
$$\gamma_{w,i,ly} = \min \left[\begin{array}{l} \exp \left[-0.693 \cdot \left(\frac{[C:N] - 25}{25} \right) \right] \\ \exp \left[-0.693 \cdot \left(\frac{[C:P] - 200}{200} \right) \right] \\ 1.0 \end{array} \right] \quad 3:1.2.8$$

where $\gamma_{w,i,ly}$ is the nutrient cycling residue composition factor for layer ly , $[C:N]$ is the C:N ratio on the residue in the soil layer, and $a_{C:P}$ is the C:P ratio on the residue in the soil layer.

Mineralization from the residue fresh organic N pool is then calculated:

$$N_{min,i,ly} = 0.8 \cdot \delta_{w,i,ly} \cdot orgN_{soil,i} \quad 3:1.2.9$$

where $N_{min,i,ly}$ is the nitrogen mineralized from the fresh organic N pool (kg N/ha), $\delta_{w,i,ly}$ is the residue decay rate constant, and $orgN_{soil,i}$ is the nitrogen in the fresh organic pool in layer ly (kg N/ha). Nitrogen mineralized from the fresh organic pool is added to the nitrate pool in the layer.



E. Mineralization and Decomposition

Implemented in *mineralization_decomp.py*.

Mineralization equations represent *net* mineralization. Both Fresh and Active N are subject to mineralization. Mineralization uses the temperature and soil water factors in calculations.

Mineralization from Active N pool (kg/ha) only occurs if the temperature of the soil is greater than 0 °C and is calculated as

$$N_{min,act} = Minrate \times (TempFac \times WaterFac)^{0.5} \times ActiveN \quad [S.4.E.1]$$

$$Minrate = Active\ N\ mineralization\ rate\ (kg/ha;\ user\ defined,\ right\ now\ at\ 0.0003)$$

N mineralized from the Active pool is transferred from the Active pool to the NH4 pool.

Decomposition and mineralization of Fresh N is only in the first soil layer. Decomposition and mineralization are a function of a daily rate constant that is calculated with the C:N ratio and C:P ratio of the residue, and temperature and soil water factors. The C:N and C:P ratios are calculated as:

$$C : N = \frac{0.58 \times res}{FreshN + NO3} \quad \text{if } FreshN + NO3 \leq 0 \quad [S.4.E.2]$$

$$C : P = \frac{0.58 \times res}{FreshOrganicP + LabileP} \quad \text{if } FreshOrganicP + LabileP \leq 0 \quad [S.4.E.3]$$

- res = above ground soil residue
- Fresh organic P = fresh organic Phosphorus in the first layer
- LabileP = labile phosphorus calculated and adjusted multiple times in phosphorus cycling

Soil P pools are tracked in the soil P model equations. A decay rate constant (Decay) defines the fraction of residue that is decomposed as:

$$Decay = MinCoeff \times ResComp \times (TempFac \times WaterFac)^{0.5} \quad [S.4.E.4]$$

$$MinCoeff = Fresh\ residue\ mineralization\ coefficient\ (0.05)$$

This decay rate constant *Decay* is the same as the decay rate used to calculate residue in Crop, Actual Growth [C.7.A.8]

Pseudocode

- Readable
- Descriptive
- Programmer Friendly
- Code Compatible

Soil Module

$$V_{olatilization} = \left[\frac{FracV_{olatil}}{FracNitr + FracV_{olatil}} \right] \times TotNitrV_{olatil}$$
$$V_{olatilization} = 0 \quad \text{if } FracNitr + FracV_{olatil} \leq 0 \quad [S.4.B.10]$$

C. N loss in leaching_runoff_erosion

Implemented in leaching_runoff_erosion.py.

All N lost in runoff and erosion is removed from soil layer 1. N in leaching is removed from a given soil layer and added to the next deeper layer.

Only soluble (inorganic) N is susceptible to loss in runoff, i.e. NO₃ and NH₄. The concentration (kg N/mm H₂O) of the inorganic pools NO₃/NH₄ in the top soil layer is:

$$NConc_1 = N \times \left[\frac{1 - \exp\left(-\frac{w}{SAT}\right)}{w} \right] \quad [S.4.C.1]$$

Where "N" is NO₃ or NH₄

w = Sum of runoff and soil water for layer (i.e. Runoff + SW)
SAT = water content at soil saturation for layer

The mass (kg/ha) of NO₃/NH₄ loss in runoff (mm) from soil layer 1 only is:

$$NRunoff = NConc_1 \times Cr \times Runoff \quad [S.4.C.2]$$

Cr = coefficient of extraction for runoff (calibrated to 0.1 for NO₃ and 1 for NH₄)
Runoff = daily runoff (mm H₂O)

Similarly, there is no inorganic nitrogen lost in erosion because it has already been accounted for in runoff. For N loss in erosion, soil N concentrations (mg/kg) for the Fresh, Active, and Stable pools are calculated:

$$NConc = \frac{100 \times N(kg/ha)}{BD \times depth} \quad [S.4.C.3]$$

BD = soil layer bulk density (g/cm³)
Depth = soil layer thickness (mm)

If Sed (daily soil loss) is 0, then ErosNLoss is 0 and ER should not be calculated. N mass loss in erosion (kg/ha) is calculated as:

$$ErosNLoss = 0.001 \times NConc \times Sed \times ER \quad [S.4.C.4]$$

The screenshot shows a GitHub issue thread. The issue title is "The spreadsheet has NH4 loss in erosion". It was created by Max Donovan on June 6 at 11:05 AM. A comment by Max Donovan on June 6 at 11:56 AM states: "I added this. In the spreadsheet, NH4Runoff is not calibrated with a coefficient of extraction". A comment by Melissa Motew on August 8 at 3:20 PM states: "Good point. The values come out to be 0 because NH4 has so little N compared to the organic pools. But, I'm not sure why there is an equation there for it to begin with." A blue circle is drawn around the entire thread.

Development

```
1 """  
2 RUFAS: Ruminant Farm Systems Model
```

```
3  
4 File name: soil_temp.py
```

```
5  
6 Author(s): William Donovan, wmdonovan@wisc.edu
```

```
7  
8 Description: This module contains the necessary functions for calculating and  
9 updating the soil temperature on a given day. Currently the only  
10 function meant to be used outside of this file is the update_all()  
11 function. The other functions are meant to serve as helper  
12 functions within this file.
```

```
13  
14 Soil attribute definitions
```

```
15  
16 Tsoil = soil temperature (°C) at depth z (mm)
```

```
17  
18 L = lag coefficient, set to 0.8
```

```
19  
20 Tsoil_prev_day = soil temperature (°C) at depth z (mm) on the previous day
```

```
21  
22 df = depth factor
```

```
23  
24 Taair = average annual air temperature (°C)
```

```
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63
```

```
64 Soil values updated by calling update_all():
```

```
65 Tsurf
```

```
66 soil_layers.temperature
```

```
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

Code

- In-line Comments
- Active Development
- Comparable Variable Names
- Pseudocode Indexing

```
84 #
85 # Calculates the soil temperature for each layer given average annual air
86 # temperature. "pseudocode_soil" S.1.A.1
87 #
88 def calc_Tsoil(soil, weather, time):
89     L = 0.8
90     # Taair = weather.T_avg_annual[time.year-1]
91     Taair = 8.18 # TODO: spreadsheet model fix. Note in pseudocode
92     dd = calc_dd(soil)
93     for x in range(len(soil.soil_layers)):
94
95         if x == 0:
96             z = soil.soil_layers[x].bottomDepth / 2
97         else:
98             z = (soil.soil_layers[x].bottomDepth +
99                 soil.soil_layers[x - 1].bottomDepth) / 2
100
101     ✗ # soil temperature (C) at depth z (mm) on previous day
102     Tsoil_prev_day = soil.soil_layers[x].temperature
103
104     # "pseudocode_soil" S.1.A.3
105     zd = z / dd
106
107     # "pseudocode_soil" S.1.A.2
108     df_exp = exp(-0.867 - 2.078 * zd)
109     df = zd / (zd + df_exp)
110
111     # "pseudocode_soil" S.1.A.1
112     Tsoil = (L * Tsoil_prev_day) + (1 - L) * \
113             (df * (Taair - soil.Tsurf) + soil.Tsurf)
114     soil.soil_layers[x].temperature = Tsoil
115
```

Equations taken from SWAT 2009 documentation. Implemented in *soil_temp.py*.

A. Base equation

$$T_{soil} = (L \times T_{soil,d-1}) + (1 - L) \times [df \times (T_{air} - T_{surf}) + T_{surf}] \quad [S.1.A.1]$$

T_{soil} = soil temperature (°C) at depth z (mm)
 L = lag coefficient, set to 0.8
 $T_{soil,d-1}$ = soil temperature (°C) at depth z (mm) on previous day
 df = depth factor
 T_{air} = Average annual air temperature (°C)
 T_{surf} = Daily soil surface temperature (°C)

$$df = \frac{zd}{[zd + \exp(-0.867 - 2.078 \times zd)]} \quad [S.1.A.2]$$

zd = ratio of depth at the center of soil layer to damping depth

$$zd = \frac{z}{dd} \quad [S.1.A.3]$$

dd = damping depth (mm)
 z = depth at the center of the soil layer

Damping depth is a function of soil water and a max damping depth as:

$$dd = dd_{max} \times \exp \left\{ \ln \left(\frac{500}{dd_{max}} \right) \times \left[\frac{(1-scale)}{(1+scale)} \right]^2 \right\} \quad [S.1.A.4]$$

dd_{max} = maximum damping depth (mm)
 $scale$ = scaling factor for soil water

$$scale = \frac{SW}{[(0.356 - 0.144 \times bd) \times z_{tot}]} \quad [S.1.A.5]$$

bd = soil bulk density (g/cm³)
 SW = total soil water in the profile (mm)
 Z_{tot} = total soil profile depth

$$dd_{max} = 1000 + \frac{(2500 \times bd)}{[bd + 686 \times \exp(-5.63 \times bd)]} \quad [S.1.A.6]$$

```
for x in range(len(soil.soil_layers)):  
    if x == 0:  
        z = soil.soil_layers[x].bottomDepth / 2  
    else:  
        z = (soil.soil_layers[x].bottomDepth +  
            soil.soil_layers[x - 1].bottomDepth) / 2  
  
    # soil temperature (C) at depth z (mm) on previous day  
    Tsoil_prev_day = soil.soil_layers[x].temperature  
  
    # "pseudocode_soil" S.1.A.3  
    zd = z / dd  
  
    # "pseudocode_soil" S.1.A.2  
    df_exp = exp(-0.867 - 2.078 * zd)  
    df = zd / (zd + df_exp)  
  
    # "pseudocode_soil" S.1.A.1  
    Tsoil = (L * Tsoil_prev_day) + (1 - L) * \  
            (df * (Tair - soil.Tsurf) + soil.Tsurf)  
    soil.soil_layers[x].temperature = Tsoil
```


Structure

Static

Dynamic

Crop Type

Maximum LAI

Minimum Base Temp



Biomass

LAI

Nutrients

Profile Depth

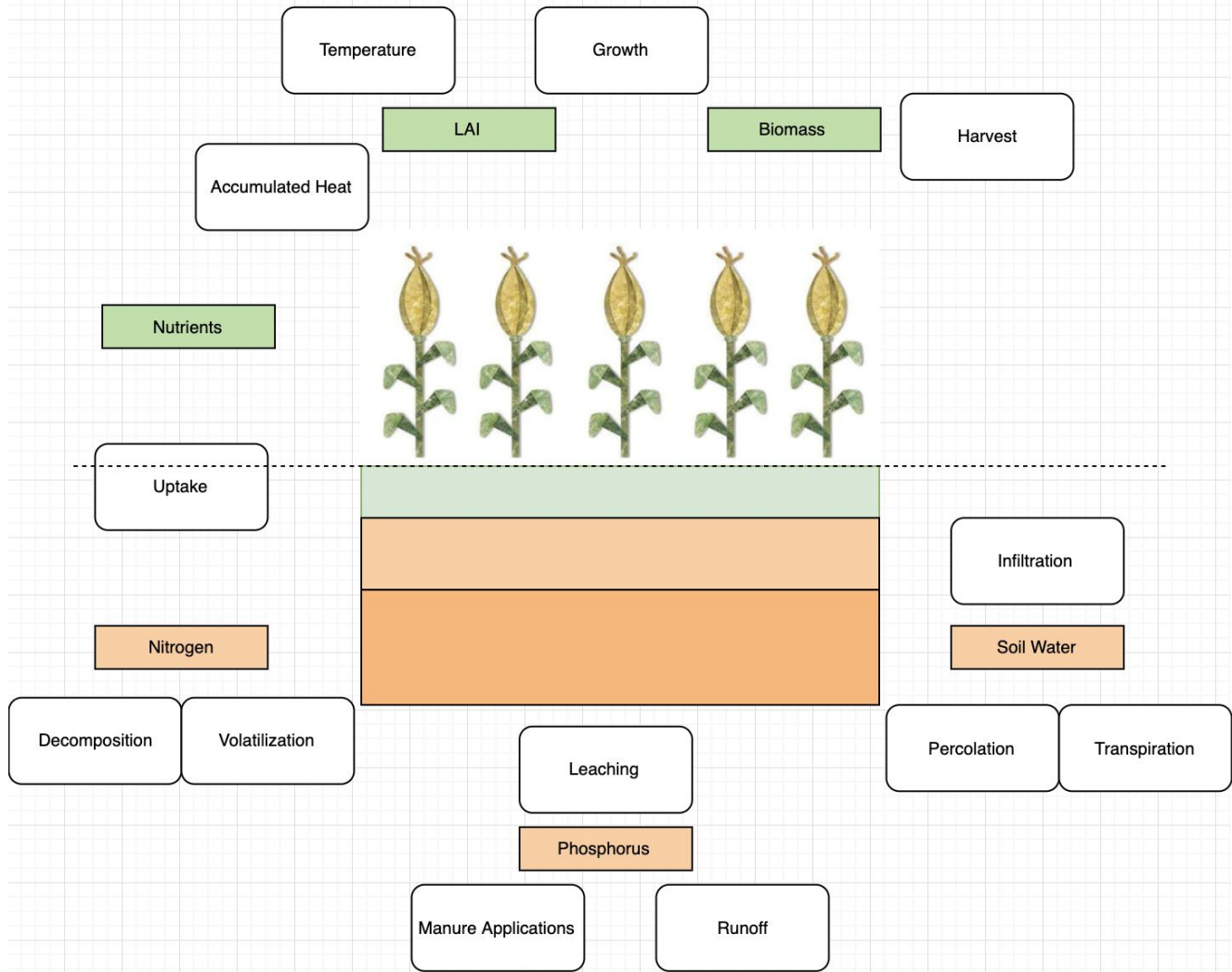
Bulk Density

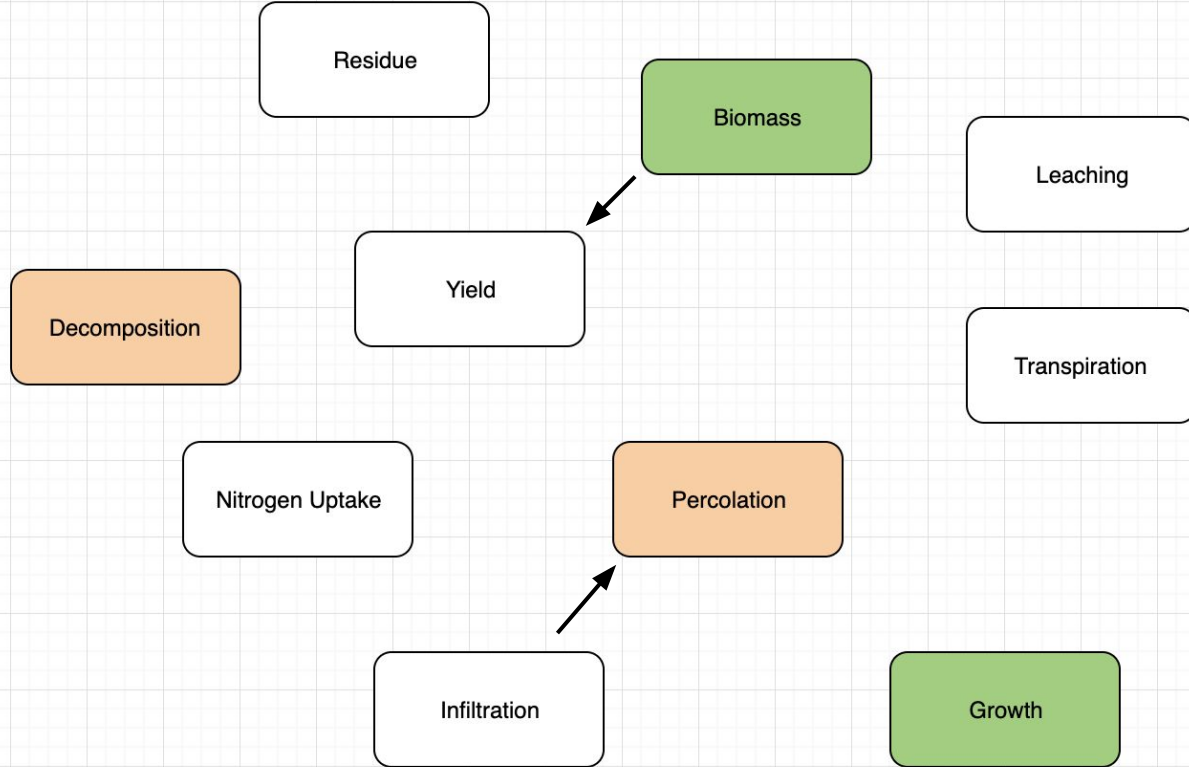
Field Capacity

Nitrogen

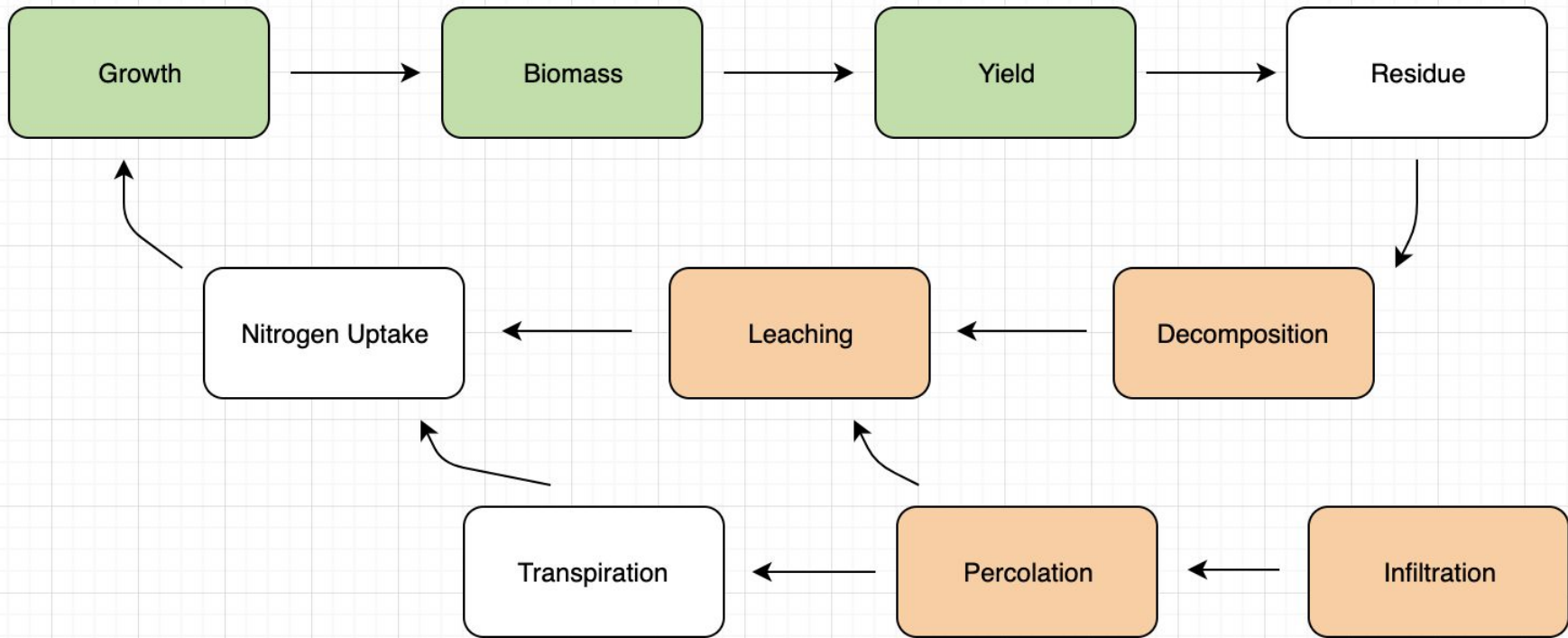
Soil Water

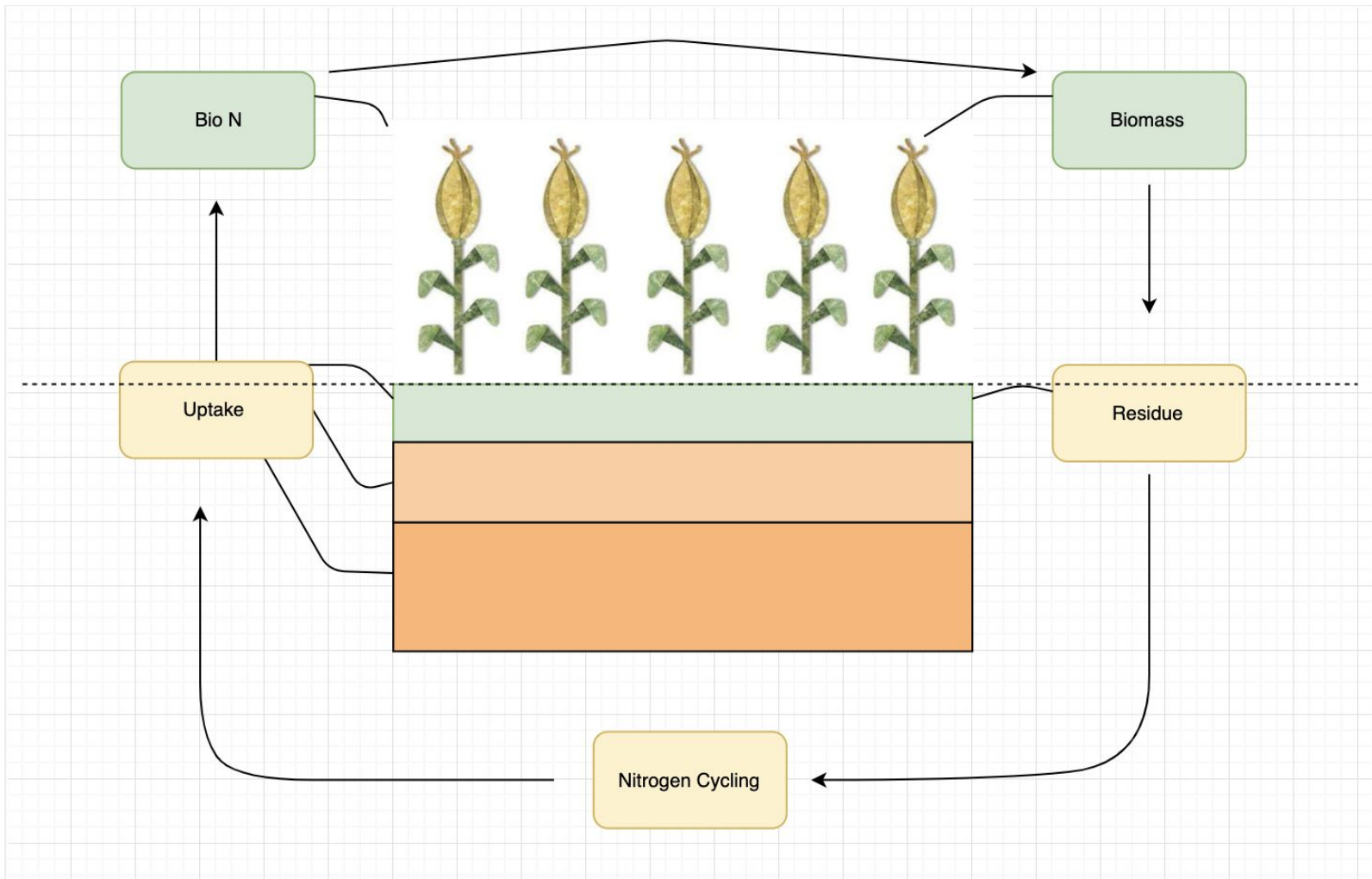
Phosphorus





Order of Operations





```
def daily_soil_routine(soil, crop, weather, time):
    """
    Description:
        Executes all the daily soil routines.

    Args:
        soil: instance of the Soil class
        crop: instance of the Crop class
        weather: instance of the Weather class
        time: instance of the Time class
    """
    # calculate and update the temperature of the soil layers
    soil_temp.update_all(soil, crop, weather, time)

    # calculate daily runoff
    infiltration.update_all(soil, weather, time)

    # calculate daily transpiration
    evapotranspiration.update_all(soil, crop, weather, time)

    # transpiration is defined in the crop module, but called here as a
    # component of water balance
    transpiration.update_all(crop.current_crop, soil, time)

    # calculate daily percolation
    percolation.update_all(soil)

    # updates daily soil water fluxes
    soil_water.update_all(soil, weather, time)

    # calculate daily soil erosion
    soil_erosion.update_all(soil, crop, weather, time)

    # calculate and update the contents of 3 organic and 2 inorganic nitrogen
    # pools
    nitrogen_cycling.update_all(soil, weather, time)
```

```
#
# This function calls all the necessary functions to update information related
# to nitrogen cycling. The order in which each method is called is significant
# and is still being worked out.
#
def update_all(soil, weather, time):
    calc_temp_factors(soil)
    calc_water_factors(soil)
    nitrification_volatilization.nitrification_volatilization(soil)
    leaching_runoff_erosion.leaching_runoff_erosion(soil)
    denitrification.denitrification(soil)
    mineralization_decomp.mineralization_decomp(soil)
    humus_mineralization.humus_mineralization(soil)
    added_manure_N(soil, weather, time)
```


Soil Temperature

Base equation

Soil Hydrology

Infiltration

Evapotranspiration

Percolation

Soil Water Update

Soil Erosion

Base equation, Rainfall Intensity, a...

Soil Nitrogen

Initialize soil N Pools

Nitrification and Volatilization

N loss in leaching_runoff_erosion

Heat Units

Available Heat

FrPHU (Fraction of Potential He...

Root Development

Root development in the soil ✕

Water Uptake

Maximum Potential Water Upta...

Impact of low soil water conten...

Actual water uptake

Nitrogen Uptake

Calculate shape coefficients for...

Plant Nitrogen Demand

Plant nitrogen uptake from soil

Phosphorus Uptake ✕

Calculate shape coefficients for...

Plant Phosphorus Demand

Plant Phosphorus uptake from ...

Growth Constraints

Water stress

Temperature stress

Nitrogen stress

Phosphorus stress

Growth Factor

Leaf Area Index

Calculate Leaf Area Index (LAI)

Biomass

Biomass

Calculate Maximum potent...

Yields

Calculate water deficiency

Potential Harvest Index

Aboveground Biomass

Actual Harvest Index

Potential Yield

Actual Yield

Nutrient Removal

Residue

Team Development

