

Walker — Waves Exam 3

April 29, 2025

TOTAL SCORE / 25

Comments and Scores for Each Problem Are on Last Page

This exam tests your fluency with the core of the Wolfram Language, as it was presented in *An Elementary Introduction to the Wolfram Language, 3rd Edition (EIWL3)*, Sections 25-34 and 38-41. There is one problem with two or three parts corresponding to each section. **Tip: all of them are meant to be quick. If you get bogged down, move on.**

Directions:

After downloading this notebook, rename it with your first name in the filename. E.g., *Eli-Exam3.nb*, *Harper-Exam3.nb*, *Hexi-Exam3.nb*, *Jeremy-Exam3.nb*, *Rania-Exam3.nb*, *Tahm-Exam3.nb*, or *Walker-Exam3.nb*.

Then disconnect from the wifi and work the exam. Save your notebook early and often so that you don't lose work in progress.

Your answers always go into the Wolfram Language Input cells that begin with a comment, e.g.,

```
(* 1a *) foobar /@ Plus[Array]
```

All your answers should execute and re-execute without warnings or error messages.

You may refer to your downloaded copies of *EIWL3*, and anything else we developed in the course (like your cheat sheets!), but not to any web resources.

When you are done, save your notebook one last time, re-join the wifi, and then email it to me.

This exam was designed to require about 45 minutes, but if you need a full hour, that is ok. Everyone will stop at the one-hour mark.

1. Applying Functions (*EIWL3* Section 25)

(a)

Use **Map** with a *levels* spec to put a frame around each individual number in the array **Array[Plus, {10, 10}]** (we don't want frames around already-framed things — just one level of frames around the individual numbers).

```
In[28]:= (* 1a *) Framed /@ Flatten@Array[Plus, {10, 10}]
```

```
Out[28]= {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
```

(b)

Copy what you did in (a), but for this part, also turn the result into a grid using **Grid** and the “as an afterthought” syntax:

```
In[29]:= (* 1b *) Grid[Array[Plus, {10, 10}], Frame -> All]
```

```
Out[29]=
```

2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18
10	11	12	13	14	15	16	17	18	19
11	12	13	14	15	16	17	18	19	20

2. Pure Anonymous Functions (EIWL3 Section 26)

(a)

Use the **#** and **&** notation to create an anonymous function that cubes whatever is given it, and then use **/@** to apply it to every member of the list **{1, 2, 3, 4, 5}**.

```
In[30]:= (* 2a *) #^3 & /@ {1, 2, 3, 4, 5}
```

```
Out[30]= {1, 8, 27, 64, 125}
```

(b)

Use the **#1**, **#2**, and **&** notation to create an anonymous function that divides its first argument by its second argument. Combine this with **Apply** and a *levelspec* to apply the function to

$\{\{1,2\}, \{2,3\}, \{3,4\}, \{4,5\}\}$. Once you have this right, you will get $\{\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}\}$.

```
In[31]:= (* 2b *) #1 / #2 &@@@ {{1,2},{2,3},{3,4},{4,5}}
```

```
Out[31]= {1/2, 2/3, 3/4, 4/5}
```

3. Applying Functions Repeatedly (EIWL3 Section 27)

(a)

Use **Nest** to apply **Factorial** twice to $\{1,2,3,4\}$. If you have this right, 620,448,401,733,239,439,360,000 will be one of the elements of your answer.

```
In[32]:= (* 3a *) Nest[Factorial, {1, 2, 3, 4}, 2]
```

```
Out[32]= {1, 2, 720, 620 448 401 733 239 439 360 000}
```

(b)

Use **NestList** to apply **Factorial** three times to $\{1,2,3\}$, as well as showing the results of doing it 0, 1, and 2 times. If you have this right, you will have an insanely large result at the third step. Do not go any higher, or I do not know what will happen to your computer.

(b)

Combine **PrimeQ** with **Select** to only list the numbers in **Range[20]** that are prime.

```
In[35]:= (* 4b *) Select[Range[20], PrimeQ]
Out[35]= {2, 3, 5, 7, 11, 13, 17, 19}
```

5. More About Pure Functions (EIWL3 Section 29)

(a)

Accomplish exactly the same thing as **Table[n*(n-1)/2, {n,6}]** using **Array** and a pure function.

```
In[36]:= (* 5a *) Array[#*{#-1}/2 &, 6] // Flatten
Out[36]= {0, 1, 3, 6, 10, 15}
```

(b)

Make some modifications to **FoldList[Plus, {1,2,3,4,5}]** so that it produces a list of the first 10 factorials. Instead of hand-coding the list up to 10, begin by first changing **{1,2,3,4,5}** to **Range[10]**.

```
In[37]:= (* 5b *) FoldList[Times, Range[10]]
Out[37]= {1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800}
```

6. Rearranging Lists (EIWL3 Section 30)

(a)

Use **Transpose** and one of the *levelspec* options to turn

{{{1,uno},{2,dos},{3,tres}},{{4,cuatro},{5,cinco},{6,seis}}} into
{{{1,2,3},{uno,dos,tres}},{{4,5,6},{cuatro,cinco,seis}}}

```
In[38]:= (* 6a *) Transpose /@ {{{1,uno},{2,dos},{3,tres}},{{4,cuatro},{5,cinco},{6,seis}}}
Out[38]= {{{1, 2, 3}, {uno, dos, tres}}, {{4, 5, 6}, {cuatro, cinco, seis}}}
```

(b)

Use **Flatten** and a *levelspec* option to turn

{{{1,uno},{2,dos},{3,tres}},{{4,cuatro},{5,cinco},{6,seis}}} into
{{1,uno},{2,dos},{3,tres},{4,cuatro},{5,cinco},{6,seis}}

```
In[39]:= (* 6b *) Flatten[{{1, uno}, {2, dos}, {3, tres}}, {{4, cuatro}, {5, cinco}, {6, seis}}, 1]
Out[39]= {{1, uno}, {2, dos}, {3, tres}, {4, cuatro}, {5, cinco}, {6, seis}}
```

7. Parts of Lists (EIWL3 Section 31)

(a)

Use the magical **All** position (you will need to use **All** more than once) to turn

{{{Eli, Lerner}, {Harper, Yonago}, {Hexi, Jin}}, {{Jeremy, Choy}, {Rania, Zaki}, {Tahm, Loyd}, {Walker, Harris}}}} into
 {{Eli, Harper, Hexi}, {Jeremy, Rania, Tahm, Walker}}

```
In[40]:= (* 7a *) {{{Eli, Lerner}, {Harper, Yonago}, {Hexi, Jin}},
               {{Jeremy, Choy}, {Rania, Zaki}, {Tahm, Loyd}, {Walker, Harris}}} [[All, All, 1]]
Out[40]= {{Eli, Harper, Hexi}, {Jeremy, Rania, Tahm, Walker}}
```

```
In[41]:=
```

(b)

Use a magical *negative positional argument* to extract {Jeremy, Rania, Tahm, Walker} from
 {{Eli, Harper, Hexi}, {Jeremy, Rania, Tahm, Walker}} and combine that with **Take** with a
 different magical *negative* argument to extract {Tahm, Walker}.

```
In[42]:= (* 7b *) {{Eli, Harper, Hexi}, {Jeremy, Rania, Tahm, Walker}} [[-1, -2;; -1]]
Out[42]= {Tahm, Walker}
```

8. Patterns (EIWL3 Section 32)

(a)

Use **Cases** to choose the lists that begin and end with the same letter in this list of lists (but look ahead to part (b) before you solve part (a)):

```
{
  {"a", "l", "u", "l", "a"},
  {"a", "l", "o", "h", "a"},
  {"a", "r", "a", "r", "a"},
  {"b", "o", "n", "u", "s"},
  {"c", "i", "v", "i", "c"},
  {"d", "e", "b", "e", "d"},
  {"e", "l", "b", "o", "w"},
}
```

```
{“z”, “a”},
{“z”, “z”}
}
```

```
In[43]:= (* 8a *) Stuff = {
  {"a", "l", "u", "l", "a"},
  {"a", "l", "o", "h", "a"},
  {"a", "r", "a", "r", "a"},
  {"b", "o", "n", "u", "s"},
  {"c", "i", "v", "i", "c"},
  {"d", "e", "b", "e", "d"},
  {"e", "l", "b", "o", "w"},
  {"z", "a"},
  {"z", "z"}
}
Cases[Stuff, {a_, ___, a_}]
```

```
Out[43]=
{{a, l, u, l, a}, {a, l, o, h, a}, {a, r, a, r, a}, {b, o, n, u, s},
 {c, i, v, i, c}, {d, e, b, e, d}, {e, l, b, o, w}, {z, a}, {z, z}}
```

```
Out[44]=
{{a, l, u, l, a}, {a, l, o, h, a}, {a, r, a, r, a}, {c, i, v, i, c}, {d, e, b, e, d}, {z, z}}
```

(b)

The pattern **BlankNullSequence** has the shorthand `___`. Use `___` to improve the pattern you used in Part (a) so that the two-letter list `{z, z}` is also included in your result.

```
In[45]:= (* 8b *) Cases[Stuff, {a_, ___, a_}]
Out[45]=
{{a, l, u, l, a}, {a, l, o, h, a}, {a, r, a, r, a}, {c, i, v, i, c}, {d, e, b, e, d}, {z, z}}
```

9. Assigning Names to Things (EIWL3 Section 38)

(a)

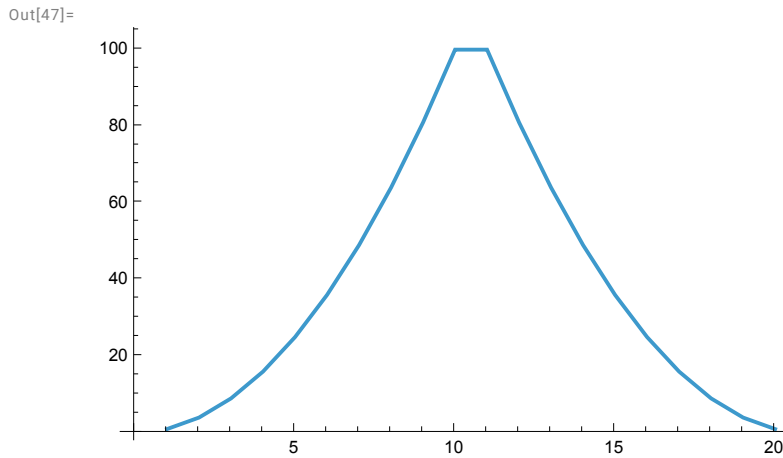
Use **Module** to compute `x=Factorial[10]`, and then produce `{x, x^2, x^3}`.

```
In[46]:= (* 9a *) Module[{x = Factorial[10]}, {x, x^2, x^3}]
Out[46]=
{3 628 800, 13 168 189 440 000, 47 784 725 839 872 000 000}
```

(b)

Inside **Module**, let `rangeSquared=Range[10]^2`, and then produce a list line plot of `rangeSquared` joined with `Reverse[rangeSquared]`.

```
In[47]:= (* 9b *) Module[{rangeSquared=Range[10]^2},
  ListLinePlot[Join[rangeSquared, Reverse[rangeSquared]]]]
```



10. Immediate and Delayed Values (EIWL3 Section 39)

(a)

Make a *one-character change* to this expression,

`Module[{x:=RandomInteger[10]}, {x, x2, x3, x4}]`, so that it produces four different powers of the same random number instead of four different powers of different random numbers.

```
In[48]:= (* 10a *) Module[{x=RandomInteger[10]}, {x, x2, x3, x4}]
```

Out[48]=
{10, 100, 1000, 10 000}

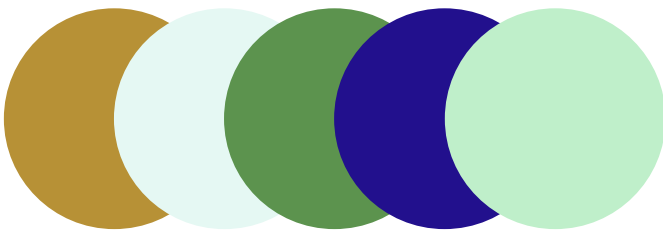
(b)

Make a *one-character change* to this expression,

`Module[{color=RandomColor[]}, Graphics[Table[Style[Disk[{i,0}], color], {i, 5}]]]`, so that it produces five different-color disks.

```
In[49]:= (* 10b *) Module[{color:=RandomColor[]},
  Graphics[Table[Style[Disk[{i,0}], color], {i, 5}]]]
```

Out[49]=



11. Defining Your Own Functions (EIWL3 Section 40)

(a)

Define a function **f** that takes a list of three elements and out of them makes a list of lists that contains all six possible orderings. Using **Permutations** will make this easy.

Include a test of your function as **f[1,2,3]** and make sure it gets
 $\{\{1,2,3\}, \{1,3,2\}, \{2,1,3\}, \{2,3,1\}, \{3,1,2\}, \{3,2,1\}\}$.

```
In[50]:= (* 11a *) f[a_, b_, c_] = Permutations[{a, b, c}]
Out[50]=
{{a, b, c}, {a, c, b}, {b, a, c}, {b, c, a}, {c, a, b}, {c, b, a}}

In[51]:= f[1,2,3]
Out[51]=
{{1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}}
```

(b)

Define a function **g** that gives **1** for **g[0]**, and gives **n*g[n-1]** for any integer **n** greater than **0**, *but don't use an If statement!* Include a test of your function as **g[6]** and make sure it gets **720**.

```
In[52]:= (* 11b *) g[0] = 1; g[n_Integer] := n * g[n - 1]
g[6]
Out[53]=
720
```

12. More About Patterns (EIWL3 Section 41)

(a)

Use the replacement rule notation — e.g., **/.** and **->** — to exchange the first and last element in any list containing two or more elements and test your replacement using the list
{alpha, beta, gamma, delta, epsilon}.

```
In[54]:= (* 12a *) Greek = {alpha, beta, gamma, delta, epsilon}
Greek /. {a_, b_., z_} -> {z, b, a}
Out[54]=
{alpha, beta, gamma, delta, epsilon}

Out[55]=
{epsilon, beta, gamma, delta, alpha}
```

(b)

Starting with **Characters/@RomanNumeral[Range[100]]**, select all the sequences correspond-

ing to the Roman numerals that have XXX in them.

```
In[56]:= (* 12b *) Cases[Characters/@RomanNumeral[Range[100]], {a___, "X", "X", "X", b___}]
Out[56]=
{{X, X, X}, {X, X, X, I}, {X, X, X, I, I}, {X, X, X, I, I, I}, {X, X, X, I, V},
 {X, X, X, V}, {X, X, X, V, I}, {X, X, X, V, I, I}, {X, X, X, V, I, I, I},
 {X, X, X, I, X}, {L, X, X, X}, {L, X, X, X, I}, {L, X, X, X, I, I},
 {L, X, X, X, I, I, I}, {L, X, X, X, I, V}, {L, X, X, X, V}, {L, X, X, X, V, I},
 {L, X, X, X, V, I, I}, {L, X, X, X, V, I, I, I}, {L, X, X, X, I, X}}
```

(c)

Use **StringJoin** to turn what you got in 12(b) into

{XXX,XXXI,XXXII,XXXIII,XXXIV,XXXV,XXXVI,XXXVII,XXXVIII,XXXIX,LXXX,LXXXI
,LXXXII,LXXXIII,LXXXIV,LXXXV,LXXXVI,LXXXVII,LXXXVIII,LXXXIX}.

```
In[57]:= (* 12c *) StringJoin@@@
Cases[Characters/@RomanNumeral[Range[100]], {a___, "X", "X", "X", b___}]
Out[57]=
{XXX, XXXI, XXXII, XXXIII, XXXIV, XXXV, XXXVI, XXXVII, XXXVIII, XXXIX, LXXX,
 LXXXI, LXXXII, LXXXIII, LXXXIV, LXXXV, LXXXVI, LXXXVII, LXXXVIII, LXXXIX}
```

1. Applying Functions 1 / 2

You ignored the directions (!) like use “Map with a levelspec ...,”
and “Copy what you did in (a),”

2. Pure Anonymous Functions 1 / 2

2(a) perfect. For 2(b) directions were again to use a levelspec, but I’m impressed with your use of @@@ to elegantly get the job done.

3. Applying Functions Repeatedly 2 / 2

Perfect.

4. Tests and Conditionals 2 / 2

Very nice.

5. More About Pure Functions 2 / 2

I love perfect answers.

6. Rearranging Lists 2 / 2

Great!

7. Parts of Lists 2 / 2

7(a) nice. 7(b) a little convoluted, and not quite what was asked for, but it worked.

8. Patterns 2 / 2

Perfect use of patterns. In 8(a) most people used __ so for them 8(b) was different.

9. Assigning Names to Things 2 / 2

Perfect use of modules.

10. Immediate and Delayed Values 2 / 2

Perfect use of immediate vs. delayed assignments.

11. Defining Your Own Functions 2 / 2

Exceedingly nice! In particular, I like that you used n_Integer in 11(b).

12. More About Patterns ___ / 3