

# Jeremy — Waves Exam 3

April 29, 2025 **TOTAL SCORE 25 / 25 Holy Smokes — Perfect — See comment on 4. 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 *levelspec* 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[135]:=

(\* 1a \*) Map[Framed, Array[Plus, {10, 10}], {2}]

Out[135]=

```
{ {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[136]:=

(\* 1b \*) Map[Framed, Array[Plus, {10, 10}], {2}] // Grid

Out[136]=

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[137]:=
(* 2a *) #^3 & /@ {1, 2, 3, 4, 5}

Out[137]=
{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[138]:=
(* 2b *) Apply[ #1 / #2 &, {{1, 2}, {2, 3}, {3, 4}, {4, 5}}, {1}]

Out[138]=
{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[139]:=
(* 3a *) Nest[Factorial, {1, 2, 3, 4}, 2]

Out[139]=
{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[142]:=
(* 4b *) Select[Range[20], PrimeQ[#] &]

Out[142]=
{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[143]:=
(* 5a *) Array[#*(#-1)/2 &, 6]

Out[143]=
{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[144]:=
(* 5b *) FoldList[Times, Range[10]]

Out[144]=
{1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800}
```

## 6. Rearranging Lists (EIWL3 Section 30)

(a)

Use **Transpose** and one of the *level/spec* 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[145]:=
(* 6a *)
Transpose[{{{1,uno},{2,dos},{3,tres}},{{4,cuatro},{5,cinco},{6,seis}}},3↔2]

Out[145]=
{{{1, 2, 3}, {uno, dos, tres}}, {{4, 5, 6}, {cuatro, cinco, seis}}}
```

(b)

Use **Flatten** and a *levels* 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[146]:=

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

Out[146]=

```
{{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[147]:=

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

Out[147]=

```
{{Eli, Harper, Hexi}, {Jeremy, Rania, Tahm, Walker}}
```

(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[148]:=

```
(* 7b *) Take[ {{Eli,Harper,Hexi},{Jeremy,Rania,Tahm,Walker}}][[-1],-2]
```

Out[148]=

```
{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[149]:=

```
(* 8a *) Cases[{
  {"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"}
}, {x_, __, x_}]
```

Out[149]=

```
{{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}}
```

(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[150]:=

```
(* 8b *) Cases[{
  {"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"}
}, {x_, ___, x_}]
```

Out[150]=

```
{{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  $x = \text{Factorial}[10]$ , and then produce  $\{x, x^2, x^3\}$ .

In[151]:=

```
(* 9a *) Module[{x = Factorial[10]}, {x, x^2, x^3}]
```

Out[151]=

```
{3 628 800, 13 168 189 440 000, 47 784 725 839 872 000 000}
```

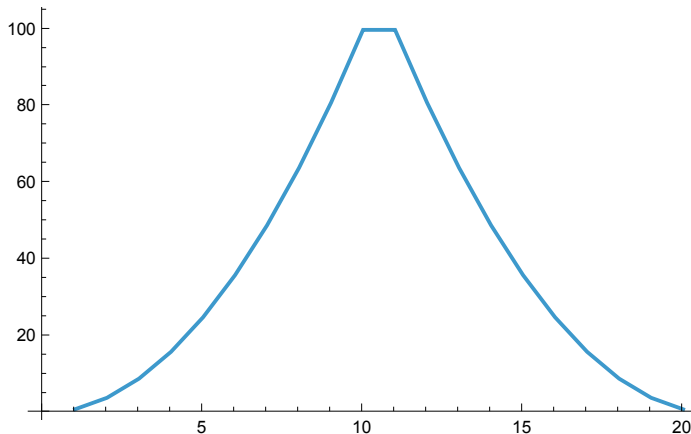
(b)

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

In[152]:=

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

Out[152]=



## 10. Immediate and Delayed Values (EIWL3 Section 39)

(a)

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

**Module** $\{x := \text{RandomInteger}[10], \{x, x^2, x^3, x^4\}\}$ , so that it produces four different powers of the same random number instead of four different powers of different random numbers.

In[153]:=

```
(* 10a *) Module[{x=RandomInteger[10]}, {x, x^2, x^3, x^4}]
```

Out[153]=

```
{4, 16, 64, 256}
```



(b)

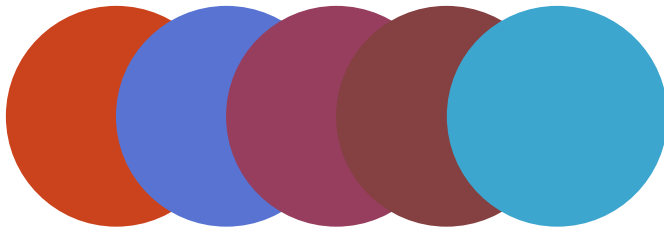
Make a *one-character change* to this expression,

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

In[154]:=

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

Out[154]=



## 11. Defining Your Own Functions (E/WL3 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[155]:=

```
(* 11a *) f[x_, y_, z_] := Permutations[{x, y, z}]
f[1, 2, 3]
```

Out[156]=

```
{{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[157]:=

```
(* 11b *) g[0] = 1; g[n_] := n * g[n - 1]
g[6]
```

Out[158]=

```
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[159]:=
(* 12a *) {alpha, beta, gamma, delta, epsilon}/.{x_,y___,z_}>{z,y,x}

Out[159]=
{epsilon, beta, gamma, delta, alpha}
```

(b)

Starting with **Characters/@RomanNumeral[Range[100]]**, select all the sequences corresponding to the Roman numerals that have XXX in them.

```
In[160]:=
(* 12b *) Select[Characters/@RomanNumeral[Range[100]],
  MatchQ[#, {___, "X", "X", "X", ___}] &]

Out[160]=
{{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[161]:=
(* 12c *) StringJoin /@
  Select[Characters/@RomanNumeral[Range[100]], MatchQ[#, {___, "X", "X", "X", ___}] &]

Out[161]=
{XXX, XXXI, XXXII, XXXIII, XXXIV, XXXV, XXXVI, XXXVII, XXXVIII, XXXIX, LXXX,
 LXXXI, LXXXII, LXXXIII, LXXXIV, LXXXV, LXXXVI, LXXXVII, LXXXVIII, LXXXIX}
```

1. Applying Functions   2   / 2

Perfect. Surprisingly (to me), only two people got this one!

2. Pure Anonymous Functions   2   / 2

Perfect. Yours are so easy to grade compared with the other wild stuff I so often get.

3. Applying Functions Repeatedly   2   / 2

Perfect.

4. Tests and Conditionals   2   / 2

Nice. Although there is a slicker form of PrimeQ that avoids PrimeQ[#]&.

5. More About Pure Functions   2   / 2

I love perfect answers.

6. Rearranging Lists   2   / 2

Really nice use of one of the levelspec options in 6(a).

7. Parts of Lists   2   / 2

Again, I love perfect answers.

8. Patterns   2   / 2

Perfect use of patterns.

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

Perfect.

12. More About Patterns   3   / 3

Perfect.