Comments on Chibany's Excellent Adventure

- (1.3) I found this statement confusing: "When every possibility is equally likely, probability is defined as the relative number of possibilities in each set."

    It's not clear what "possibilities" means. Is it events? Outcomes? Likewise in "each set," which sets is it talking about? And if probability is defined as the relative number of possibilities, what is that relative to? This might be one of those times when formulas are easier to understand than words?

- (1.3) Similarly with "Rather than each possibility counting one towards the size of a set it is in, you count the possibility according to its relative weight," but maybe if I were clear on the first one, this would also be clear?

    This set approach is cool. For some reason I always did exactly what you said and jumped straight to formulas even though the usual starting coin-toss examples are discrete. The idea of probability "mass" helped me with some intuition, and reminded me of the experiments we did in first-year calculus where you'd plot a function on paper then cut it out and weigh it to estimate the area under the curve. That kind of physicality is fun.

    As an aside, do kids still learn set theory in elementary school? My recollection is that we learned it even before arithmetic, but it was the 60's and teachers were looking for alternatives to rote learning. I wonder when sets are introduced in Japanese schools?

    Well, I poked around some on the Internet and was reminded that we learned set theory, Boolean logic, alternative bases for numbers, etc, as part of the "New Math." Personally I loved it, all the formal rules, and thinking back it probably stimulated the same part of my mind that became obsessed with proofs (especially geometry) and, later on, programming. We also wrestled with Zeno's paradoxes, which made us life-long skeptics! But I can also see why Charles Schultz expressed the frustration of parents in 1965:

    The Wikipedia article suggests that New Math died out in the 70's in the US. In Japan it was apparently part of something called the 科学化運動, a larger

movement toward emphasis on science (also perhaps over rote learning, just like in the US), which makes sense given Japan's economic direction at the time, but an article[1] suggests that in math education it was too abstract for most students, so was also abandoned in the 70's.

- (1.3) In "Yes, Chibany, it does as it always should. Your chance of getting Tonkatsu is three out of four or 0.75." maybe insert "[getting Tonkatsu] *at least once* [is...]"
- (1.3) There is also a sentence, "Note that if the possible outcomes were not equally likely, we would sum their individual probabilities to calculate the cardinality," but would you normally use "cardinality" to describe the weighted sum? I thought it would still be just an integer, the number of elements.
- (1.3) In the "See this in code" boxes it might be good to have one of those little triangular, colored toggle icons or something to indicate what people should click on. At least in Firefox the link isn't highlighted at all, so it's not clear what to click.
- (1.4) In "Chibany wants a tonkatsu dinner" there is some ambiguity that I find a bit confusing. Tanaka-san tells Chibany that at least one of the meals tomorrow will be tonkatsu, but doesn't say which one. Chibany questions whether the "second" meal will be tonkatsu, but it's not clear from context what "second" means:
    - If "second" means "dinner" then it seems like you'd get the 2/3 probability given in the explanation.
    - If "second" means "the other meal" (since Tanaka didn't specify whether the meal he was talking about was lunch or dinner), then the event of interest is {TT} and the probability of that would only be 1/3.

  so perhaps it would be good to say, in the upper paragraph, "They want to know how likely it is that dinner is a Tonkatsu."

- (1.4) Likewise in the above, if the question is dinner, then shouldn't it be HT and TT that are outlined in red?
- (1.4) Just below that we have, "In our example, we're interested in the probability of the event *A={TT}* conditioned on the knowledge that there's at least one tonkatsu, *B={HT,TH,TT}*, but if that's the event we're after, shouldn't the probability above be 1/3? It seems like we're bouncing back and forth between different interpretations.
- (1.4) Ha! Well, it turns out that Tanaka and Chibany are diving into this ambiguity on their own, without my comments!
- (pause there because I'm sleepy!!)

---

[1] https://www.researchgate.net/publication/37261895_diercidazhanhounowagaguoniokerushuxuejiaoyunofazhannitsuite-_kexuehuayundongkarashengkirushuxuehenofeixiang_-

•

Note: In what follows I should note that I'm running everything in Jupyter Lab on my own system rather than in the Colab environment.

- (2.3) It seems like there might be some kind of version dependency? When I run the first example on my Mac, it looks like `flip()` returns a Boolean rather than an integer:

```
$ python3
Python 3.13.7 (main, Aug 14 2025, 11:12:11) [Clang 17.0.0 (clang-
1700.0.13.3)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import jax
... from genjax import gen, flip
...
... @gen
... def chibany_day():
...     lunch_is_tonkatsu = flip(0.5) @ "lunch"
...     dinner_is_tonkatsu = flip(0.5) @ "dinner"
...     return (lunch_is_tonkatsu, dinner_is_tonkatsu)
...
>>> key = jax.random.key(42)
... trace = chibany_day.simulate(key, ())
... meals = trace.get_retval()
... print(f"Today's meals: {meals}")
...
Today's meals: (Array(False, dtype=bool), Array(False,
dtype=bool))
```

  It's interesting that, down below, in the section on Understanding Traces, the values returned by `trace.get_retval()` are indeed Booleans, rather than integers. Indeed for me the results are always Booleans, but there are points in the online pages where it's returning integers. I wonder what changed?

  > Ah, there's an explanation later on. Got it! It might be good to have that near the first instance where this appears, though, or people (like me) may think something is wrong.

- (2.3) Just an aside, I wonder how many students these days understand the choice of 42 as THE ANSWER?
- (2.3) In the continuation of the example, where you simulate 10,000 days of meals, before `days = jnp.stack(days_tuples, axis=1)`, you need to define `jnp`:

  ```
  import jax.numpy as jnp
  ```

  I realized later on that the reason I ran into this issue is because I clicked from (1.3) to (2.3) without going through 2.1 and 2.2, where the imports, etc, were done. The fact that people will be going through these in arbitrary order seems like quite a challenge!

- (2.3) I thought the PRNG implementation would be system-independent, but maybe not? I get a slightly different result:

  ```
  Days with tonkatsu: 7540 out of 10000
  P(at least one tonkatsu) ≈ 0.754
  ```

- (2.3) In the solution for Exercise 3: Conditional Counting, instead of just

  ```
  days = jax.vmap(run_one_day)(keys)
  ```

  it seems like you need this pattern from the earlier code:

  ```
  days_tuples = jax.vmap(run_one_day)(keys)
  days = jnp.stack(days_tuples, axis=1)
  ```

  to make the remainder of the solution work properly

- (2.3) Just an aside: when you do 100 simulations, what's the official way to collect all of the traces into a list so you can explore them later? I'm sure you could just append the output of `get_choices()` to a list, but it seems like Python supports all kinds of tricks, so I wondered if there's some clever way to do that via the @ syntax that names the choices in the first place?
- (2.5) Also, I'm not sure what can be done about this, but I realize now that I got lost. I started with the Probability tutorial, then clicked to look at the GenJax code, not realizing that I was now in a different tutorial, so I never went back! Only later did I realize these were different tutorials and that I'd skipped a bunch of background material by focusing only on the GenJax! Maybe it would help if each section were labeled? The text does talk about things like "Tutorial 1 Chapter 4" but it's not clear what those numbers refer to, so maybe in the table of contents on the left, each chapter could have those numbers? 1.1, 1.2..., 2.1, 2.2...? It might help a bit to see that I'm in Tutorial 2 rather than still in 1? I'm relabeling my comments here with that convention.

- (2.5) Interesting that you chose green and blue for the taxis. Even in the absence of color vision issues, there's a fun difference in the way Americans and Japanese draw the line between green (緑) and blue (青) hues.[2] The most obvious is the color indicating "go" in traffic signals, but also what the Japanese Tax Agency calls the "blue" form used by sole proprietors. Most English-language explanations of the form note that it isn't blue but green. It's something I guess we'll be finding out about soon! Language biases perception…
- (2.5) A detail but shouldn't P(blue|says blue)=12/(12+17) in the taxi problem round to 41.4% rather than 41.5%? It's a detail but something that might make a person wonder if they're doing the computation wrong…
- (2.6) In Step 4 it says:
    - Use `@gen` decorator
    - Name all random choices with `@ "name"`
    - Return what you want to infer
    - Use `if` statements to model dependencies

  but it seems like using `if` cause trouble in Jax world, even when you aren't using the JIT compiler. For example, this works:

```
@gen
def tricky():
    # The outcome is the identification of a taxi color
    taxi_is_blue = flip(0.15) @ "blue"
    says_blue = flip(jnp.where(taxi_is_blue,0.80,0.20)) @
"identified"
    return (taxi_is_blue,says_blue)
```

  but this generates unintelligible errors from deep inside Jax:

```
@gen
def tricky():
    # The outcome is the identification of a taxi color
    taxi_is_blue = flip(0.15) @ "blue"
    if taxi_is_blue:
        says_blue = flip(0.80) @ "identified"
    else:
        says_blue = flip(0.20) @ "identified"
    return (taxi_is_blue,says_blue)
```

  (traceback)

[2] https://en.wikipedia.org/wiki/Blue-green_distinction_in_language

```
TracerBoolConversionError: Attempted boolean conversion of traced
array with shape bool[].
The error occurred while tracing the function _inner at
/Users/tim-
burress/Documents/PC/ProbabilityTutorial/lib/python3.13/site-
packages/genjax/_src/core/compiler/interpreters/stateful.py:75
for Tracing to Jaxpr. This value became a tracer due to JAX
operations on these lines:

  operation a:bool[] = trace[
  abs_eval=<function
initial_style_bind.<locals>.bind.<locals>.wrapped.<locals>._abs_e
val at 0x122e28cc0>
  impl=<function
initial_style_bind.<locals>.bind.<locals>.wrapped.<locals>._impl
at 0x122e285e0>

in_tree=PyTreeDef((CustomNode(Const[StructStaticMetadata(child_fi
eld_names=[], static_fields={'val': 'blue'})], []),
CustomNode(genjax.flip[StructStaticMetadata(child_field_names=[],
static_fields={})], []), (*,)))
  num_consts=0
  out_tree=<function transformation_with_aux2.<locals>.<lambda>
at 0x122e293a0>
] b
    from line /Users/tim-
burress/Documents/PC/ProbabilityTutorial/lib/python3.13/site-
packages/genjax/_src/core/compiler/initial_style_primitive.py:81:
19 (initial_style_bind.<locals>.bind.<locals>.wrapped)
See
https://jax.readthedocs.io/en/latest/errors.html#jax.errors.Trace
rBoolConversionError
```

Although looking ahead to the next note it seems that this has more to do with tracing than with the use of `if`.

- (2.6) The code for Pattern 1 doesn't seem to work for similar reasons:

```
@gen
def coin_flips(n_flips, bias=0.5):
    """Generate n independent coin flips."""

    results = []
    for i in range(n_flips):
        # Each flip is independent
        result = flip(bias) @ f"flip_{i}"
        results.append(result)

    return jnp.array(results)

key = jax.random.key(42)
trace = coin_flips.simulate(key, (10, 0.7))
flips = trace.get_retval()
print(f"Flips: {flips}")
```

(traceback)

```
TracerIntegerConversionError: The __index__() method was
called on traced array with shape int32[]
The error occurred while tracing the function _inner at
/Users/tim-
burress/Documents/PC/ProbabilityTutorial/lib/python3.13/sit
e-
packages/genjax/_src/core/compiler/interpreters/stateful.py
:75 for Tracing to Jaxpr. This concrete value was not
available in Python because it depends on the value of the
argument args[0].
See
https://jax.readthedocs.io/en/latest/errors.html#jax.errors
.TracerIntegerConversionError
```

It only seems to work if you hard-code `n_flips`:

```
for i in range(10):
```

The page they give suggests some options to make this work but it's not clear how to resolve it in this case. Since the example worked for you, I wonder if there might be something I'm missing?

- (2.6) Pattern 2 defines

```
def infer_bias(k):
    trace, weight = coin_with_unknown_bias.generate(k, (10,),
observations)
    return trace.get_retval(), weight
```

but it looks in the call to `generate()` the ChoiceMap should be the second argument, rather than the third, so `(k,observations,(10,))`.

- (2.6) In Pattern 2 it looks like we need

```
from genjax import uniform
```

before defining the model. Maybe this was imported earlier, though, in the Colab environment?

- (2.6) In Pattern 2, after solving the above two issues, we run into the problem that `n_flips` is now a `DynamicJaxprTracer` rather than an `int`, so we get that weird error:

```
TracerIntegerConversionError: The __index__() method was
called on traced array with shape int32[]
The error occurred while tracing the function _inner at
/Users/tim-
burress/Documents/PC/ProbabilityTutorial/lib/python3.13/sit
e-
```

```
packages/genjax/ src/core/compiler/interpreters/stateful.py
:75 for Tracing to Jaxpr. This concrete value was not
available in Python because it depends on the value of the
argument args[0].
See
https://jax.readthedocs.io/en/latest/errors.html#jax.errors
.TracerIntegerConversionError
```

If I hard-code the argument to `range()` as `10` instead of `n_flips`

```
for i in range(10):
```

everything works, although an oddity is that even if I generate millions of samples, on MacOS (M1) the average weighted bias is converging on 0.666666..., rather than 0.70. Does that make sense? I noticed in earlier work with GenJax that there were sometimes unexpected results like this, were the answers were consistently off by some small, but non-negligible, delta. Kind of worrisome...

- (2.6) In Pattern 3, a similar issue with the position of the `ChoiceMap` argument for `generate()`:

```
def infer_weather(k):
    trace, weight = mood_model.generate(k,(),observation)
    return trace.get_retval(), weight
```

but once that is fixed, it works fine except that, oddly, my final result of running the example is 0.873 rather than 0.875.

- (2.6) In Pattern 4, the model isn't executed, so nothing can go terribly wrong, but if you do execute it

```
key = jax.random.key(42)
mm = weekly_meals.simulate(key,(14,))
```

you run into the weird typing error:

```
TracerIntegerConversionError: The __index__() method was
called on traced array with shape int32[]
The error occurred while tracing the function _inner at
/Users/tim-
burress/Documents/PC/ProbabilityTutorial/lib/python3.13/sit
e-
packages/genjax/ src/core/compiler/interpreters/stateful.py
:75 for Tracing to Jaxpr. This concrete value was not
available in Python because it depends on the value of the
argument args[0].
See
https://jax.readthedocs.io/en/latest/errors.html#jax.errors
.TracerIntegerConversionError
```

Strangely, if you don't specify a value for the input parameter `days`, and just give `generate()` an empty tuple for the arguments

```
key = jax.random.key(42)
mm = weekly_meals.simulate(key,())
```

you get a different weird error:

```
TracerBoolConversionError: Attempted boolean conversion of
traced array with shape bool[].
The error occurred while tracing the function _inner at
/Users/tim-
burress/Documents/PC/ProbabilityTutorial/lib/python3.13/sit
e-
packages/genjax/_src/core/compiler/interpreters/stateful.py
:75 for Tracing to Jaxpr. This value became a tracer due to
JAX operations on these lines:

  operation a:bool[] = trace[
  abs_eval=<function
initial_style_bind.<locals>.bind.<locals>.wrapped.<locals>.
_abs_eval at 0x12aaf4e00>
  impl=<function
initial_style_bind.<locals>.bind.<locals>.wrapped.<locals>.
_impl at 0x12aaf4ae0>

in_tree=PyTreeDef((CustomNode(Const[StructStaticMetadata(ch
ild_field_names=[], static_fields={'val': 'day_0'})], []),
CustomNode(genjax.flip[StructStaticMetadata(child_field_nam
es=[], static_fields={})], []), (*,)))
  num_consts=0
  out_tree=<function
transformation_with_aux2.<locals>.<lambda> at 0x12aaf4fe0>
] b
    from line /Users/tim-
burress/Documents/PC/ProbabilityTutorial/lib/python3.13/sit
e-
packages/genjax/_src/core/compiler/initial_style_primitive.
py:81:19
(initial_style_bind.<locals>.bind.<locals>.wrapped)

  operation a:bool[] = eq b c
    from line
/var/folders/jw/4jcvly117hzb641n8h_903k00000gq/T/ipykernel_
3371/2646648854.py:14:11 (weekly_meals)
See
https://jax.readthedocs.io/en/latest/errors.html#jax.errors
.TracerBoolConversionError
```

That last error seems to be related to the `if..else` conditional. If you replace that with the more Jax-ish:

```
current_meal = flip(jnp.where(prev_meal,0.3,0.8)) @
f"day_{day}"
```

it works fine! But only if you don't specify a value for the `days` argument. It seems weird that it would work with the default but not with anything else...

- (2.6) Pattern 5 works fine! So overall with the GenJax patterns it seems like there's only trouble when you want to pass some parameter into the generative function, because part of what that does is convert the arguments to `Tracer` objects, in which case they can no longer be used as integers, etc. Vanilla Jax has ways of dealing with that, declaring function arguments static:

```
@partial(jit, static_argnums=1)
def func(x, axis):
  return np.split(x, 2, axis)
```

  but attempts to compose `@partial` and `@gen`, or to define values for a parameter `static_argnums` in the `@gen` decorator, all seemed to fail. Obviously there must be some way to parameterize these functions, but for now it's a mystery!