Designing online quizzes and tests: Principles and implementation

Nathan Sanders

University of Toronto
Department of Linguistics
Pedagogy Coordinator

18 November 2021

Part I: Principles

Some factors to keep in mind

In general when designing a test, we want it to be:

- challenging, but fair
- quick to grade
- consistently graded
- secure against cheating

Challenging, but fair

Bloom's revised taxonomy (Anderson and Krathwohl 2001):

		Knowledge			
		factual	conceptual	procedural	metacognitive
Cognitive Process	remember	list items from a set	recognize related concepts	recall steps in a procedure	identify strategies for retaining info
	understand	summarize properties of a concept	classify items by properties	paraphrase steps in a procedure	predict personal response to an event
	apply	respond to frequently asked questions	give advice to a novice	conduct a procedure	match methods to personal strengths
	analyze	select most complete list	differentiate concepts	revise procedures given new constraints	deconstruct personal biases
	evaluate	select most effective option for a situation	determine relevance of a concept to a situation	judge effectiveness of a procedure in a situation	reflect on personal progress
	create	produce daily log	assemble team of experts	design efficient workflow	create learning portfolio

Roughly speaking, the taxonomy increases in how "challenging" the processes and knowledge are as we move downward and rightward. (*Lots* of caveats here, so don't use this uncritically.)

Challenging, but fair

We want a mix of questions from across the taxonomy (but probably not creation and metacognitive knowledge, which are a bit much for a test and more suited to bigger projects and reflective assignments).

With no challenging questions, it's hard to distinguish between the students who only know the basics and the students who have more advanced understanding beyond the basics.

With only challenging questions, it's hard to distinguish between the students who know the basics and the students who don't.

The broader the mix of questions, the less clumpy the scores will be.

Challenging, but fair

Basically, we want every student to have an opportunity to demonstrate their knowledge across (a representative portion of) all of the relevant course objectives and expectations.

Most every linguistics course has both challenging and non-challenging material, so make sure to test across all of it.

Challenging, but fair

More challenging: hypothesis comparison, applying algorithms, category tests, interpreting graphs/stats/results, making novel predictions, etc.

Less challenging: IPA symbols, definitions of technical terminology, natural classes, basic facts about important literature/theories, etc.

Challenging, but fair

Watch out for hidden expectations and biases!

For example, grammaticality judgements of standardized academic English unfairly disadvantage students who aren't native speakers of standardized varieties of English. During a test especially, there isn't enough time for them to compensate.

Re-assess whether these hidden expectations are crucial. Can we assess a student's understanding of syntax without asking them whether *who did you expect yesterday to see?* is grammatical? What do we really want to know, and how can we assess it more directly?

Quick to grade

Fast grading is important for instructors and especially TAs, who have limited hours. The less time we spend grading, the more time we can attend to actually teaching students, developing materials, etc.

Also, the faster a test is graded, the quicker students can review it and learn from it, and the sooner they can move forward in the course with a better understanding of the previous material.

This is especially important in many linguistics courses, where the course material builds upon itself week by week. Fast turnaround time is essential.

Consistently graded

Students do *not* like getting different scores for essentially the same answers. It's de-motivating and creates resentment, both of which can make them less invested in the rest of the course.

This is also a pedagogical problem for the instructor, since we can't accurately determine how the students are doing (and what material needs to be reviewed/assessed) if the grading is inconsistent.

Consistent grading also saves time later, so that we don't have to re-grade individual students' work.

Secure against cheating

We can’t prevent all cheating, so focus on minimizing cheating, by addressing the psychological reasons and the opportunities.

Students often cheat because they aren’t confident in their knowledge, so mitigate that lack of confidence.

Students often cheat because it’s easy to do, so make it harder to do.

The online environment is particularly susceptible to cheating…

Suggestions

How can we balance these considerations? Some of them are in conflict!

For example, more challenging questions are often more difficult to grade quickly and consistently.

There is no one size fits all answer here, but I've got some suggestions for the general design of a test that has worked well for me in online teaching, and it seems to do well at addressing all of the desirable features of a test.

Suggestions

Basic set-up for a quiz:

- short testing time (e.g. 30 minutes)
- large completion window (e.g. 12 hours)
- 10–15 questions
- all multiple choice or short-answer
- randomized from large pool of questions
- displayed one at a time (but backtracking is allowed)
- open book / open notes / open Quercus

Basic set-up for a midterm/final:

- 3–4 parts of the same structure as a quiz
- single large completion window (e.g. 12 hours)
- parts must be completed in order (no backtracking)

Suggestions

Short testing time combined with randomized questions combined with one question displayed at a time makes it harder to cheat. There just isn't enough time for students to work together, search through unauthorized sources, etc.

Clever use of images and links, rather than raw text, makes it even harder to cheat (they can't copy-paste as easily to a group a chat).

If the differences between questions are subtle enough, cheaters likely won't notice and thus will miss the questions anyway. Not as good as catching them, but at least they aren't benefitting.

Open book testing reduces motivation for cheating. Many cheaters cheat because they panic. If they have their notes, they are less likely to panic.

Suggestions

Multiple choice and short answer are easy to grade. Most can be autograded, so turnaround time can be brought down to zero, and there are lots of nice analytics that are more easily implemented with autograded answers.

Large completion window is very useful for addressing technical issues. If everybody has to write the test in the same 30-minute window, then it can be difficult to deal with all of the technical issues that crop up.

12-hour completion window has worked very well in an online-only environment, where students are in different time zones. I haven't tried this yet, but I expect a shorter window (6-hour or even 3-hour) should be fine for an online test in an otherwise in-person course.

Suggestions

For larger tests with multiple parts, students with testing anxiety find this structure really helpful, allowing them to focus on one part of the material at a time, taking breaks and refreshing knowledge as needed between parts.

It’s also useful for students with odd schedules, family responsibilities, etc. It’s often easier to find four separate 30-minute blocks of time than a single unbroken two-hour block.

No backtracking ever is good defence against cheating, but overly draconian. The suggested structure here (one question at a time, backtracking allowed within a quiz, no backtracking between parts of a larger test), seems to be a good compromise.

Suggestions

To make students more confident, give them some early questions that anyone who paid any attention at all could easily get. A few of these are a nice warm-up and can help lower testing anxiety,

It can also lower the defences of students who might cheat. If they see that they know the answers without cheating, they might be more inclined to keep doing the rest without cheating. If you start the test off with a hard question, they may despair and cheat immediately.

I also like to end tests with a free/easy mark as a way for the students to destress. These are also useful for adding a cushion against the lower scores that naturally come from shorter tests.

Drawbacks

Challenging questions are hard to convert into questions that are autograded. It can usually be done! But it may require a lot more upfront work than we are used to (but this work pays off in the future when the test is reused).

Online means less/no invigilation, so cheating is going to happen. Accept that it will, and work to mitigate, rather than eliminate it.

With such short testing time, we usually can't assess everything. But this is fine! You don't need to literally assess every single piece of knowledge covered in the course. The mere possibility that anything can be on the test is enough to make students study everything, Just makes sure the questions are broadly representative of the course.

Question design

Linguists are used to using “big” problems, for example, here’s a data set, list the environments of all of the phones of interest, sort them into phonemes, write the necessary phonological rules, and then write a derivation table for some particular six words.

This testing format is not amenable to such problems, and maybe that’s okay.

Big problems are hard to grade and hard to narrow down what the student knows and doesn’t know (problematic both for us and for the student), so maybe we shouldn’t use them on tests (or even on homework…).

Question design

Re-configure such problems into multiple targeted questions where the pieces are de-coupled with different data sets:

- list the environments of all of the phones of interest in a data set
- for diff data and phones, sort them into phonemes
- for diff data and allophones, write the rules
- for diff data and rules, write a derivation table

We lose the immersive aspect of solving a big problem, but we gain finer-grained insight into which specific knowledge a student has and doesn't have, and grading is faster and more consistent.

Is the immersive aspect important? Maybe for (some) homework, but usually not for tests. In my opinion, the benefits are worth losing immersion in a big problem.

Question design

But this requires more work. Not only do we have to re-think how we're used to designing problems, we have to spend the time writing up all these separate questions instead of one big problem.

Is the increased upfront workload worth it? It may not feel like it the first time around, but trust me, you will really appreciate it later on! The increased modularity makes it a lot more flexible and easier to tweak from one year to the next.

And again, the finer-grained understanding of what students know makes teaching the material more effective. If I can readily see that students struggle more with writing rules given the phonemic analysis than with writing derivation tables given the rules, then I know what needs my attention.

Question design

Open book testing requires some clever questions. There's not point in asking for the IPA symbol for a voiced labiodental fricative when students have ready access to the IPA chart. So don't ask questions like that!

Instead, ask what is the IPA symbol for a consonant with the phonation of [r], the place of [f], and the manner of [ʃ]. This can't be easily looked up: they'd have to look up four pieces of information instead of just one! This question relies on a broader understanding of the material, and is overall a better kind of question to ask (whether the test is open book or not).

Part II: Implementation with `text2qti`

text2qti

The native Canvas interface for creating quizzes on Quercus sucks, especially for questions with many multiple choice options and for large question banks.

Fortunately, there is a better solution, a Python script called `text2qti` (Poore 2020), available at `https://pypi.org/project/text2qti/`.

text2qti

`text2qti` allows you to write questions in a plain text file and convert it to the QTI format used by Quercus.

The plain text format makes it really easy to copy and paste questions to create related questions with small differences.

You can copy the entire set of multiple choice options all at once, without having to re-create each one individually, as in Quercus.

With clever use of spreadsheets, find and replace, and other tricks, you can make a huge number of questions much faster than in Quercus.

Question types

`text2qti` supports a variety of autograded question types:

- single answer multiple choice
- multiple answers (“select all”)
- mathematical questions, exact or with range
- short answer (must be exact match for one the given options)

plus a few question types that are not autograded (which I won’t cover in this workshop):

- essay questions
- file upload

Single answer multiple choice

1. Assume that hypothetical Language *X* has the words [kɛ] ‘forehead’ and [ʃɔlb] ‘night’. Based solely on universal preferences for syllable structure, which of the following is most likely to also be a word in Language *X*?

a) [mjalg] ‘dirt’
a) [zurnd] ‘beach’
a) [ɪrnv] ‘direction’
a) [sliŋ] ‘adapt’
*a) [tep] ‘nostril’
a) [θrʊ] ‘flea’

Question 1 1 pts

Assume that hypothetical Language *X* has the words [kɛ] ‘night’ and [ʃɔlb] ‘forehead’ . Based solely on universal preferences for syllable structure, which of the following is most likely to also be a word in Language *X*?

- ○ [mjalg] ‘dirt’
- ○ [zurnd] ‘beach’
- ○ [ɪrnv] ‘direction’
- ○ [sliŋ] ‘adapt’
- ○ [tep] ‘nostril’
- ○ [θrʊ] ‘flea’

Single answer multiple choice

Every question must start with a number followed by a period followed by a space. The actual number is irrelevant, so I always just use `1.`

Choices for single answer multiple choice must start with a letter followed by a right parenthesis followed by a space. The actual letter is irrelevant. I always use `a)`. The correct answer is prefixed with an asterisk * before the choice letter. Each choice must be on its own line, one after the other.

Italics can be implemented with surrounding asterisks `*  *`.

`text2qti` can handle direct Unicode input, such as IPA symbols.

Single answer multiple choice

```
1. Which of the following
signs **violates** the
syllable hand configuration
constraint?

   [sign 1](https://…)
   [sign 2](https://…)
   [sign 3](https://…)
   [sign 4](https://…)

a) sign 1
a) sign 2
*a) sign 3
a) sign 4
```

Single answer multiple choice

Questions can be larger than one paragraph. Paragraphs have to be separated by blank lines and indented by three spaces (to bring the start of each paragraph to the same point where the first paragraph starts after the initial `1.` Adjacent lines with no blank line between them will be joined up into a single paragraph.

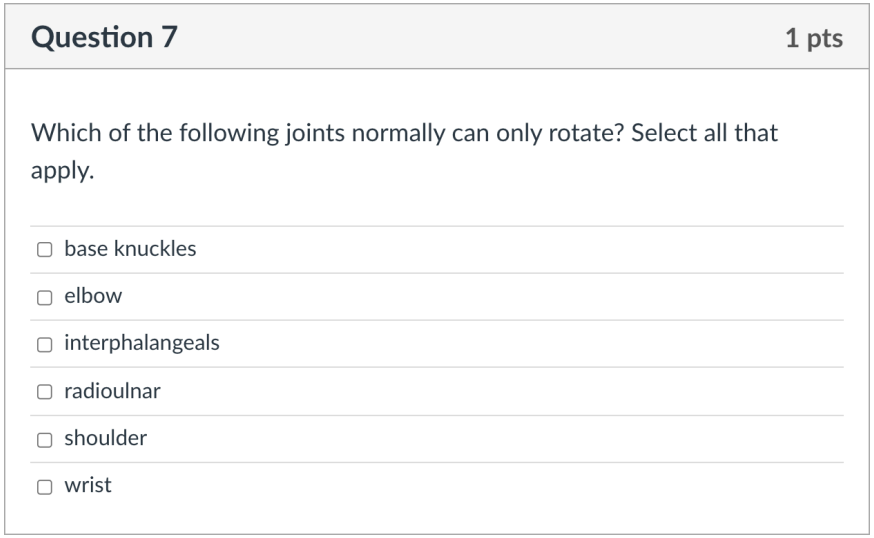
Links can be implemented with the structure `[ ]( )`, with the linked text in `[ ]` and the actual URL in `( )`.

Bold-face can be implemented with surrounding double asterisks `** **`.

Multiple answers

```
1. Which of the following
joints normally can only
rotate? Select all that apply.

[ ] base knuckles
[ ] elbow
[ ] interphalangeals
[*] radioulnar
[ ] shoulder
[ ] wrist
```



Multiple answers

Questions with multiple answers are graded in a special way, and it's good to let students know in advance. There are some number of total possible correct answers T. Students will select some number of correct answers C and some number of wrong answers W. The question itself has a total value V. A student's score is $V \times \big(C - W\big)/T$. Students get discouraged by this (even though it's a sensible system), so I recommend using these questions sparingly.

Choices for multiple answer questions must start with [ ] followed by a space. The correct answers must each have an asterisk inside the brackets: [*]. Each choice must be on its own line, one after the other.

Mathematical questions

1. Create a vector in R called **m** by running the following command:

   m <- c(44,22,49,33,59,43,21,45,54,58,48)

   Calculate the mean of **m** and enter it into the space below…

= 43.27 +- 0.01

Question 5 1 pts

Create a vector in R called **m** by running the following command:

m <- c(44,22,49,33,59,43,21,45,54,58,48)

Calculate the mean of **m** and enter it into the space below. You should enter the full unrounded output you get from R (but there is a bit of tolerance built in here, since different versions may round the output to different amounts). Do not include the preceding **[1]** in the output, just the actual mean by itself.

Mathematical questions

The answer must start with = followed by a space followed by a number. You may optionally specify a range (which is normally recommended). The range can be specified as a tolerance around the correct answer, with the target followed by +- followed by the tolerance, or instead, you can specify the range as [,], with the minimum before the comma and the maximum after the comma. The equivalent range for the question above would be encoded as:

```
= [43.26, 43.28]
```

To require an exact numerical match, just use = followed by the number:

```
= 43.27
```

Short answer

1. Consider the following data from the hypothetical language Pemol. …

   ![data231](https://…)

   What is the basic order of the subject (S), object (O), and verb (V) in Pemol?…

* 231 OVS

Question 1 1 pts

Consider the following data from the hypothetical language Pemol. You may also want to download this data and open it in another window for ease of reference for the rest of Part 3. Make note of the version number for your data (231), highlighted in yellow in the upper left. You must enter this number as part of all of your answers when you fill in a blank. Leaving this number off will result in your answer not being marked as correct by Quercus!

VERSION 231

sentence	gloss
hofo ufejid bezi	'the father stops the brother'
suvi emewid nita	'the sister needs the mother'
nita unulid hofo	'the brother follows the sister'
bezi esahid suvi	'the mother remembers the father'
bezi efejak nita	'the sister stopped the father'
suvi usahak hofo	'the brother remembered the mother'
hofo emewop suvi	'the mother will need the brother'
nita unulop bezi	'the father will follow the sister'

What is the basic order of the subject (S), object (O), and verb (V) in Pemol? Enter your response beginning with the version number of your data, followed by a space, followed by SVO, SOV, etc, all capital letters, with no other symbols or other information. For example, if your version number is 395 and the word order in Pemol is subject first, then verb second, then object last, you would enter your answer exactly as **395 SVO**

Short answer

The correct answer for short answer question must be preceded by *.

Graphics can be inserted into questions (and into options for multiple choice questions) with the code `![ ]( )`. The `[ ]` encloses the alt-text, which will be read by screenreaders, while the url for the image is in `( )`. I've never managed to get external images to work, so you can only use images uploaded to Quercus. The URL for these images as the following form:

```
https://q.utoronto.ca/courses/180273/files/11433086/preview
```

where `180273` is the course number and `11433086` is the image number (which you can find by opening it up in Quercus). It's a bit of a pain to use images, but worth it when you need them, or want to do something clever like create connected questions with randomized data (as in this example).

Short answer

1. Verbs in Pemol have an affix that indicates tense…

* 173 ap
* 148 id
* 125 ok
* 216 ap
* 157 id
* 106 ok
* 207 ap
* 164 id
* 191 ok
* 182 ap
* 231 id
* 139 ok

Question 2 1 pts

Verbs in Pemol have an affix that indicates tense (present, past, or future). What is the affix that marks the **present tense** in Pemol? Enter your response beginning with the version number of your data, followed by a space, followed by the present tense morpheme, with no other symbols or other information. For example, if your version number is 395 and the Pemol verb *tebulo* is marked as present tense with the suffix *lo*, you would enter your answer exactly as **395 lo**

Overall quiz structure

To create a quiz, you'll have a single file containing your questions. This file should start with something like the following:

```
Quiz title: M1
Quiz description: LIN 305 Midterm Part 1
```

The title is what will show up as the name of the quiz in Quercus, including in the gradebook. The description will show up in the opening textbox for the quiz.

Overall quiz structure

I like to include comments before each question or question group, so I know what the questions are asking. Comments start the line with `%`, and anything on the same line after that is ignored. Note that comments cannot be used *inside* questions.

```
% Questions 1-4: vector info
```

Overall quiz structure

If you have a single question that all students will get, that question needs to have its point value specified on the immediately preceding line, with no blank line:

```
Points: 1
1. Suppose you have hundreds of vowel duration measurements
stored in an R vector called **dur**…
```

Overall quiz structure

To create a question group, you need the following structure:

```
GROUP

pick: 2
points per question: 1

1. Suppose language *X* has the following two rules:
⋮
1. Suppose language *X* has the following two rules:
⋮
1. Suppose language *X* has the following two rules:
⋮
END_GROUP
```

Installing and running `text2qti`

First, you need to make sure your system has Python 3.6+ installed. If not, that's a bigger issue to deal with beyond the scope of this workshop. If you don't know how to do it, talk to me sometime after the workshop (or to anyone else who is computer-savvy).

Then follow the instructions here:

`https://packaging.python.org/tutorials/installing-packages/`

This will make sure you can install packages like `text2qti`. Again, if this doesn't work for you, talk to me!

If your system is set up to install packages, then download the `text2qti` files and run the following command:

```
python3 -m pip install ”text2qti”
```

Installing and running `text2qti`

Once `text2qti` is installed, you're ready to go! Create your first quiz in a convenient folder, with a suitable name, like quiz1.txt, then navigate to that folder in your command line, with something like:

```
cd /Users/nsanders/classes/LIN101/quizzes/Q1
```

Then run:

```
text2qti quiz1.txt
```

This will create a QTI file in ZIP format. In the Quercus course you want to create the quiz, go to Import Existing Content, from Content Type, select QTI .zip file, then Choose file, select your file, then finally, click Import (make sure the New Quizzes option is unchecked).

Installing and running `text2qti`

It will take a few moments for Quercus to import the quiz. Once it's been imported, you can find it in Assignments, where you go edit it like you edit any normal quiz.

And that's it!

A final piece of advice: you may find it unwieldy to have all of your questions in a single text file. I often create a separate text file for each question, and then concatenate them together before running `text2qti`. If all of the quiz question files have the same structure for the names (e.g. `quiz1-Q1.txt`, `quiz1-Q2-3.txt`, etc.), you can use a command like the following:

```
cat quiz1-Q*.txt > quiz1.txt ; text2qti quiz1.txt
```

Installing and running `text2qti`

You're going to encounter some difficulties. `text2qti` is a bit finicky, and there are some formatting requirements that you may forget about (or that I forgot to tell you!). You may have made some small error in your code.

When you make your first test, I recommend leaving your self plenty of time to do the work. Start small, work on just a few questions at a time to make sure they work. You can upload partially completed quizzes to Quercus, test them out, and delete them. As long as you don't publish them, students will never see them.

And if you give me enough notice, I can help you solve issues. Once you get used to using it, you're going to find that it's a *huge* time saver!

References I

Anderson, L. W., and D. R. Krathwohl, ed. 2001. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. New York: Longman.

Poore, Geoffrey M. 2020. text2qti, version 0.5.0. Python script, available online at `https://pypi.org/project/text2qti/`.