Appendix 1: Skill measure

Our skill measure is the score on a Python programming skills test. This test is administered by Mettl, a company that specializes in pre-hiring screening and job candidate skills assessment.[1] The test consists of two hands-on programming questions in which candidates are asked to write two programs in an online coding simulator.

Mettle offers long and short hands-on Python programming questions. The difference between long and short questions is in the complexity of the task and how much time the candidate has for completing it. Candidates are expected to complete the more complex long questions in 20 minutes and the less complex short question in 10 minutes. The test must be completed 4 hours.

How did we select the questions to include in the test?

To select the programming test questions from the Mettle test database, in June 2019 we surveyed Python programmers located in the US. We aimed to collect 15 valid observations. To achieve this goal, we surveyed 26 programmers. We dropped 11 observations because they were either not residing in the US, failed at least one attention check, had technical difficulties with filling in the survey, or did not understand all coding metrics.

In the survey, programmers read descriptions of three commonly asked long programming questions and four commonly asked short programming questions with examples. After reading those, the programmers ranked the long and short questions in terms of how useful *"knowing a job candidate test score is for predicting their Python coding skill"*. The test consists of the long programming question with the lowest average rank and the short programming question with the lowest average rank.

With this procedure, we selected the following two questions. For the first question (long question) the candidate has to write a program that adds up the largest row sum and the largest column sum from an N-rows*M-columns array of numbers. For the second question (short question) the candidate has to write a program that can determine whether characters in the first string can be rearranged to form characters in the second string.

Which programming metrics do we use to evaluate candidates' test performance?

Programming skill consists of many dimensions. After conversations with several industry and academic experts, we decided on the following five metrics:

[1]Mettl is one of the largest and fastest growing online talent measurement solution providers globally. The company has been at the forefront of online assessment technology since its inception in 2010. It is assisting over 1,500 global companies, 24 Sector Skill Councils and 15 educational institutes across 80+ countries.

1) Test cases score

This score measures if the program does what it is supposed to do. Each of the two questions are evaluated by testing whether they successfully complete 10 test cases. The more test cases the program completes, the higher the score.

The test cases score is taken from the Mettl online platform. Mettl gives 1.5 points for each successfully completed test case for the long question and 0.5 points for each successfully completed test case for the short question. Mettl's test case score therefore ranges from 0 to 20. We have rescaled the test cases score to be between 0 and 75 for the long programming question and 0 and 25 for the short programming question (keeping the weighting of the two test scores implied by Mettl). Our total test cases score can range between 0 and 100.

2) Efficiency score

This score measures how efficient or fast the program is. The efficiency score is based on how long it takes the programs to complete each test case. These completion times are measured by Mettl and displayed on the online platform. As our benchmark for runtime, we will first identify the benchmark candidate, that is, the candidate with the lowest average overall run time who has completed all 10 test cases. Second, for each test case we will compare the run time of the benchmark candidate relative to the candidate. This will be done separately for the long question and the short question. We will assign 7.5 points (for the long question) and 2.5 points (for the short question) for each test case for which the candidate's code executes the test case faster than the benchmark code or up to 300 milliseconds slower for the long questions and up to 200 milliseconds slower for the short questions. We will assign 0 points for each test case that is more than 300 millieseconds slower (for long the long question) or more than 200 milliseconds slower (for the short question) or did not successfully complete the test case. The total efficiency score can range between 0 and 100.[2]

3) Complexity score

This score measures the complexity of the code. Less complex code is generally considered better because it is easier to read and maintain.

We calculate this score with the linter Pylint (version 2.4). Linters are programs that analyze code to flag programming errors, stylistic error, and suspicious constructs. In particular, we let the linter run over the codes for the long and short question so that it can calculate the McCabe's Cyclomatic complexity score.

To make sure that the complexity score can be between 0 and 100, we will take the set of codes that successfully completed all test cases (or most test cases in case no code completes

[2] The benchmark candidate receives the maximum possible score.

all of them) and assign the value of 75 for the long question and 25 for the short question with the lowest McCabe's Cyclomatic complexity score.

To penalize more complex code, we subtract 12 points (for the long question) and 4 points (for the short question) for each additional point on the cyclomatic complexity score.

Example

> Imagine there are 75 pieces of code that successfully completed all 10 test cases for the long question.
>
> The lowest McCabe's Cyclomatic complexity score of all of these questions is 4.
>
> We assign each code for the long questions that has a McCabe's Cyclomatic complexity score of 4 the complexity score of 75.
>
> Codes that have McCabe's Cyclomatic complexity scores of 5 and 6.5 will receive scores of 63 (75 – 1*12) and 57 (75-1.5*12).

To avoid rewarding code that is less complex because it is unfinished, we assign a complexity score of 0 to code that did not successfully complete any test cases.

4) Coding convention score

This score aims to measure whether the code follows coding conventions as outlined in the style guide for Python code PEP 8.

The score is calculated using Pylint version 2.4.[3] To calculate the coding convention score, we will use the standard coding convention score calculated by Pylint. The formula for this score is as follows:

> 10.0 - ((float(5 * Frequency of convention errors) / Number of statement) * 10).

The score has a maximum of 10 but no lower bound. As is common with this score, we set a minimum value of zero (i.e., negative values are censored at zero). The short question is then multiplied by 2.5 to give a maximum score of 25 while the score from the long answer question is multiplied by 7.5 to give a maximum score of 75.

To avoid rewarding code that has a low coding convention score because it is unfinished, we assign a coding convention score of 0 to code that did not successfully complete any test cases.

5) Frequency of errors score

This score aims to measure whether the code contains programming errors and bugs, for example, when break or continue keywords are used outside a loop or when a class has

[3] The list of coding convention errors can be found here https://pylint.readthedocs.io/en/latest/technical_reference/features.html

duplicate bases.[4] To calculate the errors score, we use the standard coding error score calculated by Pylint. The formula for this score is as follows:

10.0 - ((float(5 * Frequency of Errors) / Number of statement) * 10)

The score has a maximum of 10 but no lower bound. As is common with this score, we set a minimum value of zero (i.e., negative values are censored at zero). The short question is then multiplied by 2.5 to give a maximum score of 25 while the score from the long answer question is multiplied by 7.5 to give a maximum score of 75.

To avoid rewarding code that has a low frequency of errors score because it is unfinished, we assign a frequency of errors score of 0 to code that did not successfully complete any test cases.

How did we determine the weights of these measures?

We determined the weights of these measures with the help of the responses of the above-mentioned programmers' survey. In this survey, we described each of the five measures and asked if the respondent understood it. We asked each respondent to "*decide how much each score should be weighted so that the final score for the candidate is the best measure of their Python coding skill"*. By taking the average answers of the 15 respondents, we determined the following overall test weights:

Score	**Weight**
Test cases	28.53%
Efficiency	21.40%
Complexity	21.07%
Coding convention	11.53%
Frequency of errors	17.47%

The resulting formula to calculate the Python coding skill measure (i.e the final Python test score) is:

Python skill measure = test cases score*.2853 + efficiency score *.2140 + complexity score*.2107 + coding convention score*.1153 + frequency of errors score*.1747

[4] The list of coding errors can be found here https://pylint.readthedocs.io/en/latest/technical_reference/features.html