Generowanie zadań z pythonem

https://jacadzaca.github.io/blog/generating_a_maths_worksheet.html

27 lutego 2022

Inspiracja

Inspiracja

- irytacja połączona z empatią

Inspiracja

- irytacja połączona z empatią
- Khan Academy

Inspiracja

- irytacja połączona z empatią
- Khan Academy
- Podobne programy:

Inspiracja

- irytacja połączona z empatią
- Khan Academy
- Podobne programy:
  - `https://www.education.com/worksheet-generator`

Inspiracja

- irytacja połączona z empatią
- Khan Academy
- Podobne programy:
  - `https://www.education.com/worksheet-generator`
  - `https://mathsbot.com/generators/textbook`

Inspiracja

- irytacja połączona z empatią
- Khan Academy
- Podobne programy:
  - `https://www.education.com/worksheet-generator`
  - `https://mathsbot.com/generators/textbook`
  - `https://www.wolframalpha.com/problem-generator`

Jaka powinna być lista zadań?

Jaka powinna być lista zadań?

1. całosciowa, dogłębna i różnorodna

Jaka powinna być lista zadań?

1. całosciowa, dogłębna i różnorodna
2. *decorum*

Jaka powinna być lista zadań?

1. całosciowa, dogłębna i różnorodna
2. *decorum*
3. przenośna (drukowalna)

Jaka powinna być lista zadań?

1. całosciowa, dogłębna i różnorodna
2. *decorum*
3. przenośna (drukowalna)
4. odpowiedzi na ostatniej stronie

Jak?

Jak?

- zestawy zadań? kombinacje znaczków

Jak?

- zestawy zadań? kombinacje znaczków
  - itertools

Jak?

- zestawy zadań? kombinacje znaczków
  - itertools
  - random

Jak?

- zestawy zadań? kombinacje znaczków
  - itertools
  - random
- jinja+LaTeX (realizacja punkt 3)

Jak?

- zestawy zadań? kombinacje znaczków
  - itertools
  - random
- jinja+LaTeX (realizacja punkt 3)
- sympy (realizacja punkt 4)

Przykładowy sposób implementacji

Przykładowy sposób implementacji

Wymyślimy i zaimplementujemy generator zadań na roziwązywanie równań kwadratowych

Przykładowy sposób implementacji

Wymagania?

Przykładowy sposób implementacji

- rozróżniać równania kwadratowe na podstawie liczby rozwiązań (realizacja punktu 1)

Przykładowy sposób implementacji

- rozróżniać równania kwadratowe na podstawie liczby rozwiązań (realizacja punktu 1)
- współczynniki równania nie mogą być egzotyczne (realizacja punktu 2)

Przykładowy sposób implementacji

- rozróżniać równania kwadratowe na podstawie liczby rozwiązań (realizacja punktu 1)
- współczynniki równania nie mogą być egzotyczne (realizacja punktu 2)
- łatwiejsze, potem trudniejsze

- rozbicie na 3 funkcje

- rozbicie na 3 funkcje
- współczynniki powinny mieć maksymalną wartóść

- rozbicie na 3 funkcje
- współczynniki powinny mieć maksymalną wartóść
- znaki? łatwiej wygenerować najpierw wartości, a potem wybrać im znak

- rozbicie na 3 funkcje
- współczynniki powinny mieć maksymalną wartóść
- znaki? łatwiej wygenerować najpierw wartości, a potem wybrać im znak
- $\Delta > 0 \Leftrightarrow$ dwa rozwiązania

- rozbicie na 3 funkcje
- współczynniki powinny mieć maksymalną wartość
- znaki? łatwiej wygenerować najpierw wartości, a potem wybrać im znak
- $\Delta > 0 \Leftrightarrow$ dwa rozwiązania
- $\frac{b^2}{4a} > c$ szukamy ograniczenia górnego na a i c, względem b. Przyjmijmy $c > 2$, zatem $\frac{b^2}{8} > a$

- rozbicie na 3 funkcje
- współczynniki powinny mieć maksymalną wartość
- znaki? łatwiej wygenerować najpierw wartości, a potem wybrać im znak
- $\Delta > 0 \Leftrightarrow$ dwa rozwiązania
- $\frac{b^2}{4a} > c$ szukamy ograniczenia górnego na a i c, względem b. Przyjmijmy $c > 2$, zatem $\frac{b^2}{8} > a$
- $\Delta = 0 \Leftrightarrow$ jedno rozwiązanie

- rozbicie na 3 funkcje
- współczynniki powinny mieć maksymalną wartość
- znaki? łatwiej wygenerować najpierw wartości, a potem wybrać im znak
- $\Delta > 0 \Leftrightarrow$ dwa rozwiązania
- $\frac{b^2}{4a} > c$ szukamy ograniczenia górnego na a i c, względem b. Przyjmijmy $c > 2$, zatem $\frac{b^2}{8} > a$
- $\Delta = 0 \Leftrightarrow$ jedno rozwiązanie
- możemy wybrać a i c, wtedy $b = \sqrt{4ac}$

- rozbicie na 3 funkcje
- współczynniki powinny mieć maksymalną wartość
- znaki? łatwiej wygenerować najpierw wartości, a potem wybrać im znak
- $\Delta > 0 \Leftrightarrow$ dwa rozwiązania
- $\frac{b^2}{4a} > c$ szukamy ograniczenia górnego na a i c, względem b. Przyjmijmy $c > 2$, zatem $\frac{b^2}{8} > a$
- $\Delta = 0 \Leftrightarrow$ jedno rozwiązanie
- możemy wybrać a i c, wtedy $b = \sqrt{4ac}$
- aby upewnic się, że b będzie "przyjemny", niech a i c są kwadratami

- $\Delta < 0 \Leftrightarrow$ zero rozwiązań

- $\Delta < 0 \Leftrightarrow$ zero rozwiązań
- Mamy ograniczenie na b: $b < \sqrt{4ac}$, jeżeli $ac > 0 \wedge b > 0$ (znak b możemy wybrać potem)

```
>>> a, c = sign * random.randint(1, max_coefficient), sign * random.randint(1, max_coefficient)
>>> b = math.sqrt(random.randint(1, math.floor(math.sqrt(4*a*c))))
>>> b
2.23606797749979
>>> import sympy
>>> sympy.Rational(str(b))
223606797749979/100000000000000
>>> sympy.Rational(str(math.trunc(b*10)/10))
11/5
>>>
```

- $\Delta < 0 \Leftrightarrow$ zero rozwiązań
- Mamy ograniczenie na b: $b < \sqrt{4ac}$, jeżeli $ac > 0 \wedge b > 0$ (znak b możemy wybrać potem)

- $\Delta < 0 \Leftrightarrow$ zero rozwiązań
- Mamy ograniczenie na b: $b < \sqrt{4ac}$, jeżeli $ac > 0 \wedge b > 0$ (znak b możemy wybrać potem)
- "obetnijmy" ułamek; $b = \frac{\lfloor b \cdot 10 \rfloor}{10}$

Przetłumaczmy na kod

```python
import math
import sympy
import random

SIGN = [-1, 1]

def generate_quadratic_two_solutions(max_coefficient=20):
    b = random.randint(3, max_coefficient)
    a = random.randint(1, (b**2)//8)
    c = random.randint(1, (b**2)//(4*a))
    return sympy.sympify(f'{a}*x**2 + {b}*x + {c}')

def generate_quadratic_one_solution(max_coefficient=20):
    sign = random.choice(SIGN)
    a, c = sign * random.randint(1, max_coefficient)**2, sign * random.randint(1, max_coefficient)**2
    b = int(math.sqrt(4*a*c))
    return sympy.sympify(f'{a}*x**2 + {b}*x + {c}')

def generate_quadratic_zero_solutions(max_coefficient=20):
    sign = random.choice(SIGN)
    a, c = sign * random.randint(1, max_coefficient), sign * random.randint(1, max_coefficient)
    b = math.sqrt(random.randint(1, math.floor(math.sqrt(4*a*c))))
    b = sympy.Rational(str(math.trunc(b * 10)/10))
    return sympy.sympify(f'{a}*x**2 + {b}*x + {c}')
```

```python
def quadratic_difficulty_comperator(x):
    return sum(x.as_coefficients_dict().values())
```

sympy.printing.latex

```
\documentclass{article}

\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{multicol}
\usepackage[inline]{enumitem}
\usepackage[left=10mm, right=0mm, top=15mm, bottom=15mm]{geometry}

\sloppy
\begin{document}
\small
\raggedright
{% for task in tasks %}
\begin{multicols}{2}
\section{ {{ task.instruction }} }
    \begin{enumerate}[label={(\roman*)}]
        {% for problem in task.problems%}
            \item $f(x) = {{ problem }}$,
        {% endfor %}
    \end{enumerate}
\end{multicols}
{% endfor %}

\newgeometry{margin=0mm}
\newpage

\tiny
\begin{enumerate}
\begin{multicols}{3}
\setlength{\columnsep}{-10cm}
{% for task in tasks %}
    \item \begin{enumerate*}[label={(\roman*)}]
        {% for awnser in task.awnsers %}
            \item ${{ awnser }}$,
        {% endfor %}
    \end{enumerate*}
{% endfor %}
\end{multicols}
\end{enumerate}
\end{document}
```

```python
#!/usr/bin/env python3
import math
import random
import collections

import sympy
from jinja2 import Environment, FileSystemLoader

#...quadratic generating code ommited for breviety...#


Task = collections.namedtuple('Task', ['instruction', 'problems', 'awnsers'])

ENV = Environment(
    loader=FileSystemLoader('.'),
    trim_blocks=True,
    lstrip_blocks=True)

def main():
    # generate diffrent kinds of quadratic equations
    zero_solutions_quadratics = [quadratics.generate_quadratic_zero_solutions() for _ in range(10)]
    one_solution_quadratics = [quadratics.generate_quadratic_one_solution() for _ in range(10)]
    two_solutions_quadratics = [quadratics.generate_quadratic_two_solutions() for _ in range(6)]
    # sort them according to 'difficulty' - the greater the sum of quadratic's coefficients, the more difficult it is
    zero_solutions_quadratics.sort(key=quadratics.quadratic_difficulty_comperator)
    one_solution_quadratics.sort(key=quadratics.quadratic_difficulty_comperator)
    two_solutions_quadratics.sort(key=quadratics.quadratic_difficulty_comperator)

    # arrange the problems in random order, but try to keep the problems' difficulty incremental
    problem_lists = [zero_solutions_quadratics, one_solution_quadratics, two_solutions_quadratics]
    problems = []
    for _ in range(len(zero_solutions_quadratics) + len(one_solution_quadratics) + len(two_solutions_quadratics)):
        choice = random.choice(problem_lists)
        problems.append(choice.pop())
        if not choice:
            problem_lists.remove(choice)

    # generate the awnsers with sympy https://docs.sympy.org/latest/index.html
    awnsers = (sympy.printing.latex((sympy.solvers.solveset(quadratic, domain=sympy.S.Reals))) for quadratic in problems)
    # convert into LaTeX
    problems = map(sympy.printing.latex, problems)

    #output latex code
    latex = ENV.get_template('problem_sheet.jinja.tex').render(tasks=[Task('Find the roots of function $f$, given by
expression:', problems, awnsers)])
    print(latex)

if __name__ == '__main__':
    main()
```

Cały kod:

https://github.com/jacadzaca/zadanko

Demo :D

Pytania?