Tartalomjegyzék

Tájékoztató

Ez az oldal a Házifeladat Ellen?rz? rendszer használatát írja le, hogy hogyan lehet feladatokat feladni, felhasználókat és kurzusokat kezelni.

Belépés

A rendszer maga egy leibniz-es felhasználón keresztül érhet? el:

```
hazi@leibniz.math.bme.hu
```

Ide be lehet ssh-zni, scp-zni vagy a webmail-jébe belépni.

Honlap

Mivel a `hazi` egy sima felhasználó a leibniz-en, van honlapja is:

http://math.bme.hu/~hazi

Ide lehet közérdek? (publikus) infókat kitenni, de lehet jelszóval védett al-oldalakat is csinálni, mondjuk azok biztonsága elég enyhe.

Érdekesség, hogy vannak terhelési grafikonok is.

Mappaszerkezet

Ennek a felhasználónak a `home` mappájában a következ?ket találjuk:

```
[~]
????[hazijavitorendszer]
?   ????[HW]
?   ?   ????[feladat]
?   ?   .
?   ?   . (többi feladat)
?   ?   .
?   ?   ????main
?   ?   ????validate
?   ?   ????userinfo.tsv
?   ?   ????auxiliary.py
?   ?   ????off
?   ?   ????feladattipus1
?   ?   ????feladattipus2
?   ?   ????...
?   ?   ????getsenderinfo
?   ????mailsend-go_1.0.7_linux-64bit.deb
?   ????Dockerfile
?   ????Dockerfile.test
```

```
????[solution]
????[logs]
????[archive]
????digest_logs.sh
????checkpoints.sh
????...
????archive.sh
????run.sh
```

Mit csináljunk

Hallgatók kezelése

Ezt lényegében egy `hazijavitorendszer/HW/*.tsv` (tab-separated-values) fájl szerkesztésével tehetjük meg.

Formátuma:

```
email   name    course
borbely@math.bme.hu     Gábor Borbély   info2,lecturer
```

- Több ilyen fájl is lehet, ekkor a rendszer lényegében uniózza ezen fájlokat.
- Minden sora egy felhasználó
- A felhasználók az email-címükkel vannak azonosítva, a nevük csak tájékoztató jelleg?.
- Egy felhasználóhoz megadhatunk `course`-t, ami az általa látogatott kurzusok listája: egy vessz?vel elválasztott lista.
- Egy kurzus neve csak latin alfanumerikus karakterekb?l állhat (szóköz, vessz?, ékezetes karakter nem lehet benne) vagy alulvonásból (azaz regex `\w+`).
    - A példában meg van adva egy `lecturer` csoport is a tanároknak.
- Ha több kurzusra is jár egy hallgató, akkor a kurzust egy vessz?vel vagy szóközzel elválasztott listával adjuk meg.
- Ha egy hallgató többször szerepel a listában, akkor csak a legels? el?fordulását vesszük figyelembe.
- Régi hallgatókat nem érdemes kitörölni, ha a kurzusa évszámmal is meg van jelölve.
    - például a kurzus `info2_2019` nem fog összeakadni az `info2_2020` kurzussal
    - de ha valaki másodszorra hallgatja az info2-t, akkor lehet két kurzusa: `info2_2019,info2_2020`

Ha a felhasználók `.tsv` fájlját változtatjuk, akkor a módosítások csak akkor jutnak érvényre, ha újra `build`-eljük a `docker image`-et

Feladatok felvétele

Egy feladatot a `hazijavitorendszer/HW/` mappában lév? mappa definiál.

- a mappa neve a feladat neve
- almappákat nem vesz figyelembe a rendszer, ezen belül nem lehet más almappa
- kell legyen egy `manifest.json` fájl a feladat mappájában
- kellenek tesztek a mappában
    - minden teszt neve **`i`** bet?vel kell kezd?djön
    - a tesztek formátuma feladattípustól függ. lásd lentebb

Például a `fahrenheit` nev? feladat felvételéhez hozzuk létre az alábbiakat:

```
HW
????fahrenheit
    ????manifest.json
    ????i1.json
    ????i2.json
    ????i3.json
```

Egy feladathoz 123-nál több teszt esetet nem adhatunk meg!

Ha egy új feladatot felveszünk vagy régit módosítunk, vagy kitörlünk, akkor a módosítások csak akkor jutnak érvényre, ha újra `build`-eljük a `docker image`-et

manifest.json

A feladatot egy **json dictionary** írja le, az alábbi kulcsokkal:

- **"http://wiki.math.bme.hutype"http://wiki.math.bme.hu** a feladat típusa, kell legyen egy, a típussal egyez? nev?, futtatható fájl a `HW` mappában
- **"http://wiki.math.bme.hudescription"http://wiki.math.bme.hu** HTML source, json escaped, opcionális
- **"http://wiki.math.bme.hucourse"http://wiki.math.bme.hu** mely csoportok küldhetnek be (lecturer-t mindig érdemes belevenni)
    - ez lehet egy sztring, amiben vessz?vel elválasztva vannak a kurzusok
    - vagy lehet egy json lista:

```
"http://wiki.math.bme.hucourse"http://wiki.math.bme.hu: ["http://wiki.math.bme.huinfo2"http://wiki
```

ha nem adunk meg kurzust, akkor senkit?l nem fogad el beküldéseket

- **"http://wiki.math.bme.huvisible"http://wiki.math.bme.hu** `true` vagy `false`
    - ha `false` akkor ez a feladat nem fogad el beküldéseket
- **"http://wiki.math.bme.hudeadline"http://wiki.math.bme.hu** egy sztring, ami a határid?t írja le
    - például `"http://wiki.math.bme.hu2020-02-14 01:00:00 UTC+1"http://wiki.math.bme.hu`
    - **Muszáj id?zónát megadni**, ha UTC id?zónában adjuk meg, akkor is tegyünk egy **+0**-t az id?pont mögé.
    - a python dateutils.parser.parse függvénye számára értelmezhet? formátumban kell legyen
    - Ha nincsen egyáltalán "http://wiki.math.bme.hudeadline"http://wiki.math.bme.hu kulcs a szótárban, akkor bármikor be lehet küldeni a feladatot.
- **"http://wiki.math.bme.hudisclaimer"http://wiki.math.bme.hu**
    - Ezzel megkövetelhetünk egy adott formátumú levéltörzset a beküld?t?l.
    - Használható arra, hogy muszáj legyen beírni a hallgatónak azt, hogy ? készítette a feladatot és nem másolt.
    - Az értéke egy sztring kell legyen, amiben az alábbi behelyettesítéseket is megkövetelhetjük:
        - ◊ {name}
        - ◊ {email}
        - ◊ {course}
    - Például:

```
"http://wiki.math.bme.hudisclaimer"http://wiki.math.bme.hu: "http://wiki.math.bme.huÉn, {name}, fe
```

- A disclaimer többnyelv? is lehet, ha egy json listában több ilyen sztringet is megadunk.
    - Ekkor az számít helyes beküldésnek, ha a levél törzse a megadott disclaimer-ek legalább egyikével megegyezik.

    - De egyszer?en megadhatunk üres disclaimer-t is, aminek a következménye, hogy csak üres levelet fogad el feladat.
    - Ha nincsen disclaimer kulcs a szótárban, akkor a levél törzse irreleváns.
- **"http://wiki.math.bme.huresponse"http://wiki.math.bme.hu**
    - Ezzel állíthatjuk, hogy a beküld? mit kapjon meg válaszként.
    - az értéke az alábbiak egyike (string-ként) vagy ezek listája:
        - "http://wiki.math.bme.hudescription"http://wiki.math.bme.hu az eredmény mellé megkapja a feladat kiírását is. A feladat leírását egy speciális emaillel is meg lehet szerezni, szóval a beküldés után annyira nincsen szükség magára a feladatra, de érdemes ezt is beletenni a válaszemail-be.
        - "http://wiki.math.bme.huscore"http://wiki.math.bme.hu az elért pontszám, ha lehet, akkor azt is odaírja hogy mennyib?l, de az egy feladatra kapható maximális pontszám nem jól definiált.
        - "http://wiki.math.bme.hutests"http://wiki.math.bme.hu a kiértékel? script kimenete, ha ezt megadjuk és a kiértékelést végz? kód kiírja hogy melyik teszt sikerült, akkor kvázi az elért pontszámot os elárultuk.
    - Ha nem adunk meg "http://wiki.math.bme.huresponse"http://wiki.math.bme.hu mez?t a manifest-be, akkor a válasz mindent információt tartalmazni fog.

Feladattípusok

Egy feladat típusa határozza meg, hogy milyen programnyelvet várunk el a beküld?t?l. Akkor tekinthet? valami egy értelmes feladattípusnak, ha van egy olyan nev? futtatható fájl a `HW` mappában. Például ha programozási feladatból python3 programokat akarunk feladni, akkor kell legyen egy `python3_program` nev? futtatható állomány. A továbbiakban ez ellen?rzi le azt a feladatot, aminek a "http://wiki.math.bme.hutype"http://wiki.math.bme.hu mez?jében a "http://wiki.math.bme.hupython3_program"http://wiki.math.bme.hu-ot adtuk meg.

Lentebb részletezzünk, hogy milyen feladattípusok vannak és hogy melyik milyen sajátosságokkal rendelkezik. De definiálhatunk saját feladattípust is.

program

Általában bármilyen parancssorból hívható programot tesztelhetünk ezzel. Akkor is ha interpretált vagy ha fordított nyelven van írva. Persze ehhez installálva kell legyen a szükséges fordító és/vagy interpreter a Docker image-ben.

Akkor használjunk ilyen faladattípust, ha

- azt akarjuk, hogy a beküldés egy interpreter által futtatott vagy fordító által fordított fájl legyen, ami parancssorból m?ködik

Ahhoz hogy egy ilyen programot teszteljünk, az alábbiakat kell megadnunk a `manifest.json` fájlban:

- `"http://wiki.math.bme.hutype"http://wiki.math.bme.hu: "http://wiki.math.bme.huprogram"http://wiki.math.bme.hu`
- `"http://wiki.math.bme.hucompile"http://wiki.math.bme.hu` ennek az értéke egy parancs docker exec formátumban, vagy ilyenek listája.
    - ez fog el?ször lefutni, a beküldött program el?tt
    - például: `["http://wiki.math.bme.hugcc"http://wiki.math.bme.hu, "http://wiki.math.bme.hu-o"http://wiki.math.bme.hu, "http://wiki.math.bme.humyprogram"http://wiki.math.bme.hu,`

"http://wiki.math.bme.humyprogram.c"http://wiki.math.bme.hu]
    - ♦ vagy ha több lépést szeretnénk:
      [["http://wiki.math.bme.hucmake"http://wiki.math.bme.hu,
      "http://wiki.math.bme.hu."http://wiki.math.bme.hu],
      ["http://wiki.math.bme.humake"http://wiki.math.bme.hu]]
    - ♦ ha ez nincsen megadva, vagy üres lista van megadva, akkor nincsen fordítási lépés.
    - ♦ Ha bármelyik fordítási lépés **nem-nulla hibakód**ot ad, akkor a tesztek le sem futnak.
- "http://wiki.math.bme.hucommand"http://wiki.math.bme.hu ez fog lefutni tesztenként
    - ♦ lehet egy string vagy string-ek listája
        - ◊ ha egy string akkor az a futtatható állomány fog lefutni, a tesztekt?l függ? parancssori argumentumokkal
        - ◊ ha string-ek listája, akkor docker exec formátumban értend?, plusz esetleges parancssori argumentumok
    - ♦ például:
      "http://wiki.math.bme.hu./myprogram"http://wiki.math.bme.hu, ha el?z?leg lefordítottuk
    - ♦ vagy ["http://wiki.math.bme.hupython3"http://wiki.math.bme.hu,
      "http://wiki.math.bme.hufahrenheit.py"http://wiki.math.bme.hu]
    - ♦ vagy ["http://wiki.math.bme.huwolfram"http://wiki.math.bme.hu,
      "http://wiki.math.bme.hu-script"http://wiki.math.bme.hu,
      "http://wiki.math.bme.hucalculate.m"http://wiki.math.bme.hu]
- Ezek a parancsok mind a beküld? felhasználójának home-mappájában (/home/dummy) fognak lefutni és a dummy felhasználó nevében (és jogosultságaival).
- Ezen kívül az általános description, deadline, ... mez?k is lehetnek.

Egy teszt esethez egy **i** bet?vel kezd?d? nev? json fájlt kell berakni a feladat mappájába. Például ioverscrupulous.json:

```
{
   "http://wiki.math.bme.huargv"http://wiki.math.bme.hu:  ["http://wiki.math.bme.hu1.text"http://w
   "http://wiki.math.bme.hufile"http://wiki.math.bme.hu: ["http://wiki.math.bme.hu1.text"http://wi
   "http://wiki.math.bme.hureturncode"http://wiki.math.bme.hu: 0,
   "http://wiki.math.bme.hustdout"http://wiki.math.bme.hu: "http://wiki.math.bme.hu314\n240\1729"h
}
```

Ennek kulcsai:

- bemenet
    - ♦ argv
        - ◊ például:
          "http://wiki.math.bme.huargv"http://wiki.math.bme.hu:
          ["http://wiki.math.bme.hua"http://wiki.math.bme.hu,
          "http://wiki.math.bme.hu-h"http://wiki.math.bme.hu,
          "http://wiki.math.bme.hufile.txt"http://wiki.math.bme.hu]
        - ◊ ezek a futtatandó parancs mögé append-álódnak
    - ♦ bemeneti fájlok, amiket olvashat a beküld? program a futása során
        - ◊ egy string, vagy string-ek listája.
        - ◊ ezeknek a fájloknak a nevei a feladat mappájától relatívak
        - ◊ bemásolódnak az adott teszt el?tt a beküldött program mellé
        - ◊ példa:
          "http://wiki.math.bme.hufile"http://wiki.math.bme.hu:
          "http://wiki.math.bme.huinput.txt"http://wiki.math.bme.hu
    - ♦ stdin, mit kapjon a standard bemeneten, egy string

◊ például:
`"http://wiki.math.bme.hustdin"http://wiki.math.bme.hu:`
`"http://wiki.math.bme.hua\nb\n10\n"http://wiki.math.bme.hu`

- kimenet
  - ♦ stdout, egy string, hogy mit várunk az stdout-ra.
    - ◊ ha nincsen megadva, akkor mindegy, hogy mit írt ki az stdout-ra.
  - ♦ stderr, egy string, hogy mit várunk az stderr-ra.
    - ◊ ha nincsen megadva, akkor mindegy, hogy mit írt ki az stderr-re.
  - ♦ returncode: milyen return code-al kell hogy megálljon a program
    - ◊ ha nincsen megadva, akkor mindegy, hogy milyen hibakóddal lépett ki.

Opcionálisan megadhatunk egy (python3-ban írt) `test.py` nev? ellen?rz? script-et, ami mindezen információk birtokában eldöntheti, hogy elfogadja-e a megoldást.

```
def _eval(_input, stdout, stderr, returncode):
```

Ennek True/False-t kell visszaadnia. Bemenete:

- `_input`, lényegében a teszt json fájl, python dictionary-ként.
- stdout: a program által írt kimenet, string
- stderr: a program által írt hiba-kimenet, string
- returncode: a program által visszaadott hibakód, int

python3type

Akkor használjunk ilyen faladattípust, ha

- python3 kódot szeretnénk kérni a megoldásban
- például egy függvény vagy osztály megírása a feladat
- nem számít hogy mit ír ki a beküld? az stdout és stderr-re, hanem valamilyen megadott kódnak kell a megfelel? visszatérési értéket adnia.

A `manifest.json` fájl tartalma

- `"http://wiki.math.bme.hutype"http://wiki.math.bme.hu:`
  `"http://wiki.math.bme.hupython3type"http://wiki.math.bme.hu`
- `"http://wiki.math.bme.hucompile"http://wiki.math.bme.hu` opcionális, lásd a program feladattípusnál
- `"http://wiki.math.bme.hucode"http://wiki.math.bme.hu` mely kód fusson le, aminek a kimenetét akarjuk ellen?rizni
  - ♦ ha nincs megadva, akkor a következ?:

```
def _code(_input):
    return EXERCISE(*_input)
```

ahol `EXERCISE` helyett a feladat neve szerepel.

Ahogy eddig is, a feladat mappájába egy `test.py` fájlba rakhatunk saját ellen?rz? kódot. S?t itt olyan osztályokat is definiálhatunk, melyek nincsenek beépítve, de a megoldáshoz elengedhetetlenek. A kiértékel? kód alapértelmezésben:

```
def _eval(_input, _output, _expected_output, _exception, _expected_exception):
    return type(_output) == type(_expected_output) and \
           _output == _expected_output and \
```

```
type(_exception) == type(_expected_exception)
```

Az `_eval` függvény paraméterei:

- **_input** a bemeneti pickle fájlból betöltött objektum
- **_output** a `_code` függvény által return-ölt objektum
  - vagy `None`, ha a függvény id?közben Exception-t dobott
- **_expected_output** a teszt által elvárt kimenet (az adott `"http://wiki.math.bme.huo*.pkl"http://wiki.math.bme.hu` fájlból betöltött objektum)
- **_exception** a függvény futása közben dobott kivétel
  - vagy `None` ha nem volt Exception
- **_expected_exception** a teszt által elvárt kivétel, ha a teszt lényege az, hogy kivétel dobódjon
  - Ezt egy **e** bet?vel kezd?d? pickle fájlban tudjuk megadni, aminek a nevének többi része megegyezik a hozzá tartozó bemenetével.
  - Ha nincs ilyen fájl, akkor ez `None` lesz.

Ha az `_eval` függvény Exception-t dob, akkor az a teszt hibás lesz, de a kivételnek nem íródik ki a traceback-je, csak maga a kivétel. Ez azért fontos, mert ez más kategóriába esik, mintha a beküld? kódja dob kivételt, ami lehet elvárt is. Meg abból a szempontból is más ez a kivétel, hogy olyan kódban történt, ami *priviliged*-ként fut, ezért nem érdemes nagyon hangoztatni. Elvileg a hibaüzenet tartalmazhatja is a kívánt megoldást.

függvény

Ezt a feladattípust használhatjuk egy függvény megírására. Tegyük fel, hogy a feladat neve `abc` és egy pont ilyen nev? függvényt akarunk megíratni, aminem két paramétere van és a kett? összege a kimenete.

```
def abc(a, b):
    return a + b
    # ez már a megoldás
```

Ekkor a bemeneti és kimeneti pickle fájlokba valami ilyesmit kell tenni:

- 0-adik teszt
  - `i0.pkl: (0, 0)`
  - `o0.pkl: 0`
- 1. teszt
  - `i1.pkl: ('a', 'b')`
  - `o1.pkl: 'ab'`

osztály

Ezzel a feladattípussal feladhatjuk egy osztály megírását is.

- Legyen például a megírandó osztály neve `Class`, nem kell, hogy a feladat nevével megegyezzen.
- Amit megkövetelünk ett?l az osztálytól:
  - konstruktora kapjon egy paramétert (a self-en kívül)
  - tárolja el a kapott paramétert egy `x` nev? adattagban.
  - Ha nem egész számot kapott, emeljen `ValueError` kivételt

Ehhez a következ? legyen a `_code` függvény (`"http://wiki.math.bme.hucode"http://wiki.math.bme.hu` mez? a `manifest.json`-ban):

```
def _code(_input):
    return Class(_input)
```

A teszt pickle-ök pedig az alábbiak:

- 0-adik teszt
  - `i0.pkl`: `3`
  - `o0.pkl`: `{'x': 3}`
- 1. teszt
  - `i1.pkl`: `'a'`
  - `o1.pkl`: `None`
  - `e1.pkl`: `ValueError()`

És ami a legfontosabb, a `test.py`:

```
class Class:
    pass

def _eval(_input, _output, _expected_output, _exception, _expected_exception):
    return type(_expected_exception) == type(_exception) and type(_output) == Class and _output.__
```

Ez a következ? képen fog m?ködni.

- Ha beküld? nem definiált `Class` osztályt egy egyparaméteres konstruktorral, akkor a `_code` függvény hibát fog dobni, és nem `ValueError` típusút. Ez az `_eval` függvény ellen?rzésén fenn fog akadni.
- Ha a beküld? definiált egy `Class` egyváltozós *függvényt*, de az nem egy `Class` típusú példányt ad vissza (nem is tudna), akkor a `_eval` függvény ellen?rzésén fennakad.
- Ha a beküld? definiált egy `Class` osztályt a megfelel? konstruktorral, akkor a `_code` függvény egy példányt ad vissza, ami beleíródik a megfelel? output pickle fájlba (ha más kivétel nem volt).
  - Ezután az `_eval` függvény akkor tudja megkapni ezt az objektumot, ha a `test.py`-ban definiálva van egy `Class` prototípus.
  - Figyelem, mindegy hogy milyen (nem-statikus) tagváltozók vagy metódusok vannak az osztályban, mert a beküld? által létrehozott objektum lesz benne, nem az általunk megírt
  - S?t a `test.py` fájlban nincs is példányosítás!
  - a pickle-nek elég ha létezik az a típus, mindegy hogy milyen metódusokkal, mert a példány adattagjait visszaolvasáskor nem a konstruktorral tölti fel, hanem máshogyan.
- Így az `eval` függvény már le tuja ellen?rizni, hogy a kívánt típusú-e az objektum és hogy a kívánt adattagokat tartalmazza-e.

Figyelem, a `test.py`-ba ne oldjuk meg a feladatot, viszont érdemes ide egy **konstruktort és egy `__repr__` metódust** tenni. Technikailag a tesztek és a beküldések m?ködni fognak akkor is, ha csak egy `pass` van az osztályban, mert ha a beküld? megírta rendesen az osztályt és a `_code`-ban az példányosult, akkor az már elég.

Figyelem, az `_eval` függvénybe ne tegyünk tagfüggvény hívást, se lépdányosítást! Ha ezt megtesszük, akkor annak az lesz a következménye, hogy a `test.py`-ban megírt kód fog lefutni, nem pedig a beküld? kódja!

Viszont a `_code` függvényben a beküld? kódja fog futni, viszont ide se tegyük be a megoldást, mert ezt kvázi láthatja a beküld? (stack-trace-el).

text

Ez a feladattípus egy szövegfájlt vár beküldésnek, amit aztán tetszésünk szerint értelmezhetünk.

- A manifest.json fájlba az általános mez?kön kívül nem lehet mást megadni.
- A feladat mellé rakhatunk egy (python3-ban írt) `test.py` fájlt is, ami leellen?rzi a feladatot.
- A feladat mellé rakhatunk egy `solution.txt` fájlt is, ami a mintamegoldást tartalmazza.
- A feladat kiértékelése úgy történik, hogy a felhasználó által feltöltött fájl tartalmát összeveti a minta megoldással (ha van).
- Ezt az `_eval` függvény végzi, mit a `test.py`-ban felüldefiniálhatunk, az alapértelmezett kiértékel?:

```
def _eval(_reference_text, _submitted_text):
    return 1 if _reference_text.strip() == _submitted_text.strip() else 0
```

Ennek a függvénynek egy nem-negatív egész számot kell visszaadnia, az lesz a kapott pontszám.

felelet választós

Ezzel a feladattípussal lehet feleletválasztós kérdést is feltenni. Például:

1. Melyik a helyes?
    - A: 1=0
    - B: 1 != 0
    - C: 1 < 0
2. Válaszd ki a legkisebb terület?t:
    - A: egység sugarú kör
    - B: egy oldalhosszú négyzet
    - C: Az x^2 grafikonja és az y=0, x=0 és x=1 egyenesek által közrefogott síkidom.

- Ekkor a válasz egy 2-soros szövegfájl, aminek mindegyik sora az ABC bet?k valamelyike (pontosan az egyike)
- A minta beküldés (`solution.txt`):

```
B
C
```

- A kiértékel? függvény pedig a következ?:

```
def _eval(_reference_text, _submitted_text):
    submission = list(_submitted_text.strip().split())
    solution = list(_reference_text.strip().split())
    if len(submission) != len(solution):
        print("Number of answers is {} but it should be {}!".format(len(submission), len(solution)
        return 0
    return sum([submission[i] == solution[i] for i in range(len(submission))])
```

A kapott pontszám pedig 0-2-ig fog terjedni.

build

Ezzel a paranccsal lehet *élesbe* helyezni a rendszert.

```
cd ~ && docker build -f hazijavitorendszer/Dockerfile -t hazicp hazijavitorendszer
```

- Ha ezt megtesszük, akkor a legközelebbi beküldés a build-elés pillanatában meglév? állapotokat fogja látni.
- Amíg ezt nem tesszük meg, addig bármi lehet a `hazijavitorendszer` mappában, nem lesz hatással a hallgatók beküldéseire.
- Figyelem, a `hazijavitorendszer` mappán kívüli fájlok/mappák módosítása ellen ez nem véd!

definiálás

Hogyan definiálhatunk új feladattípust? (Csak "http://wiki.math.bme.huexpert"http://wiki.math.bme.hu-eknek!)

Mit NE csináljunk

chmod

- A `hazi` felhasználó `umask`-ja 027, ezt ne változtassuk!
- Minden mappa és fájl jogosultsága olyan, hogy `other` felhasználók ne lássák, ezt ne változtassuk!

script-ek

- A felhasználók táblázat és a feladatok mappáinak kivételével semmihez ne nyúljunk.
    - Csak "http://wiki.math.bme.huexpert"http://wiki.math.bme.hu-eknek!
- Ezalatt értem a home-mappa tartalmát, illetve a HW mappában a script-eket.

Logs

A rendszer folyamatosan meg?rzi és eltárolja a beküldések legfontosabb adatait. De magát az eredeti beküldést csak a levelez?rendszer INBOX-ában tudjuk megnézni.

log fájlok

A `logs` mappában van minden log, amit a rendszer generált, ezek csak a beküldések kivonatai:

- ki, mikor küldött be, melyik feladatot
- Ha *valid* volt a beküldés (kurzus, határid?, beküld? mind rendben volt), akkor a pontszámát is
- Ha egy beküldés nem volt *valid*, de bizonyos gyenge követelményeket teljesített (pl. csak elkésett a beküldéssel), akkor a rendszer kiértékeli a feladatát, de nem ad rá pontot. Ezeket is láthatjuk a log-okban.

A log fájlok-ból ha valaki csak a pontokra kíváncsi, akkor a `<SUCCESS>` szóra kell `grep`-elni és az elért maximumot kikeresni.

Itt a második példában a beküldést kiértékelte a rendszer, de INVALID címkével (mondjuk úgyis 0 pontos lett volna):

```
[2020-02-17 13:42:47 UTC] <SUCCESS> submission from "http://wiki.math.bme.huborbely@math.bme.hu"ht
[2020-02-17 13:42:48 UTC] INVALID submission from "http://wiki.math.bme.huborbely@math.bme.hu"http
```

Ezekb?l a log-okból bizonyos segéd script-ekkel tudjuk kinyerni a pontokat.

archivált

Minden feladatról, minden beküld?r?l számon van tartva az *adott pontszámot elért legutolsó* beküldése. Ez úgy történik, hogy a rendszer (egészen pontosan az `archive.sh` script) mindig eltárolja a legutolsó beküldést, olyan fájlnévvel, ami tartalmazza a feladatot, a beküld?t és az elért pontszámot.

Ezt megnézhetjük az **`archive`** mappában.

```
[archive]
????[feladat1]
?   ????jozsi~1.py
?   ????sanyi~2.py
?   ????jozsi~2.py
?   ...
?
????[feladat2]
????[feladat3]
...
```

Így mindig megkereshetjük a legutolsó beküldést, vagy a legjobb beküldést is. Vagy mondjuk csak a maximális pontot elért legutolsó beküldést.

Segéd script-ek

checksum

Mivel az egész rendszer igen érzékeny minden benne lév? script-re, ezért van egy md5 checksum, ami teszteli, hogy nem írtunk-e bele véletlenül valamelyik fontos fájlba.

Ez a checksum kiszámolódik minden (interaktív) belépésnél, ezt látjuk itt:

```
Using username "http://wiki.math.bme.huhazi"http://wiki.math.bme.hu.
hazi@leibniz.math.bme.hu's password:

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Feb 20 14:30:24 2020 from 152.66.166.49
f30a1e390a073b0f650ecfc6f0233ebf  -
hazi@leibniz:~$
```

Build-elés el?tt érdemes leellen?rizni, hogy még mindig ugyan az-e ez a checksum, mint ami a belépésnél volt. Ezt megtehetjük úgy is, hogy meghívjuk a `checksums.sh` bash script-et:

```
hazi@leibniz:~$ bash checksums.sh
f30a1e390a073b0f650ecfc6f0233ebf  -
hazi@leibniz:~$ bash checksums.sh -l
```

- Ennek a script-nek van egy `-l` opciója, ami nemcsak az összes fájlra vett MD5 hash-t írja ki, hanem fájlonként is.
- A szükséges fájlokról Borbély Gábornak (`borbely`) van egy mentése. Ha valamit elrontunk, akkor t?le vissza lehet nyerni a helyes fájlokat.
- Ez a hash érték nem érzékeny (többek között) az alábbiakra:
    - a feladatokra és a lefuttatandó tesztekre, azokat mindenki elronthatja saját felel?sségére (és kárára).
    - A log-okra és korábbi beküldésekre, pontok állására

- a `.tsv` fájlok tartalmára, amiben a felhasználók vannak
- A fentebb felsorol adatokról nincs is *hivatalos* mentés, ezeket minden feladatkit?z?nek magának kell meg?riznie.

Pontok

```
bash checkpoints.sh "http://wiki.math.bme.huemail"http://wiki.math.bme.hu "http://wiki.math.bme.hu
```

Ez kiírja az adott (email címmel definiált) felhasználó adott feladatának pontszámát.

- Ha a felhasználó üres (`"http://wiki.math.bme.hu"http://wiki.math.bme.hu`), akkor az adott feladat összes beküld?jének a pontját írja ki
- Ha a feladat üres (`"http://wiki.math.bme.hu"http://wiki.math.bme.hu`), akkor az adott ember összes beküldésének pontját írja ki
- Ha mindegyik üres, akkor minden ember minden beküldésének max pontját írja ki.
- Ez a script nem nézi az elkésett vagy érvénytelen beküldéseket, csak a minden beküldési feltételnek megfelel? feladatok pontszámát veszi figyelembe.
- Lehet a pontokat az utolsó vagy a maximum szerint nézni
  - `-m` vagy `--max`, ez a default: legjobb érvényes beküldés
  - `-l` vagy `--latest`, a legutolsó, de még id?ben beküldött, eredményt nézi

run.sh

Ha a home mappából futtatjuk a **run.sh** script-et, akkor lehet szimulálni egy beküldést.

- Ehhez meg kell adni a script-nek, hogy mely fájlokat és milyen levelet küldött egy (fiktív) beküld?.

```
bash run.sh -h
```

- Ez a script fut le akkor is, amikor valaki egy valódi levelet küld.
- Ha megadjuk a `--test` kapcsolót az elején, akkor hasonló történik, csak
  - a válaszlevél nem elküld?dik, hanem kiíródik a konzolra
  - Nincsen log-olás
  - Nincsen a megoldás kitörölve az ellen?rzés után (azért hogy újra lehessen tesztelni ugyanazzal a fájllal)
  - Nincsen archiválás
  - Mindig újra `build`-el?dik a `docker image`, vagyis nem kell manuálisan megtennünk és nincs is hatással az *éles* beküldésekre
- Ezzel érdemes kísérletezgetni, ha valaki új feladatokon dolgozik

kapcsolók

- teszt mód
  - `--test`
  - `-t`
- help
  - `--help`
  - `-h`
- a további argumentumok vagy egyetlen mappanév, vagy (egy vagy több) fájlnév
  - Ha mappanevet adunk meg, akkor a mappában lév? összes fájlt a beküldés részének tekinti.
  - Ha fájlt vagy fájlokat, akkor azon fájlokat tekinti a beküldés csatolmányainak

beküldés

- Figyelem, muszáj a beküldéshez legalább egy, kiterjesztés nélküli `info` nev? fájlt megadni, ami az (imitált) email adatait tartalmazza
- Ha egy tényleges levél érkezik, akkor ezt a fájlt a levelez?rendszernek kell szolgáltatnia (ahogyan a csatolmányok letöltésér?l is gondoskodik).
- Az `info` fájl formátuma: legalább három soros utf8 kódolású szövegfájl
    1. sora a levél beküld?je
        - ◊ Figyelem, a **tényleges beküld?** nem feltétlenül a levél **From** mez?je, azt könny? meghamisítani
    2. sora a levél tárgya
    3. sora a levél megérkezésének dátuma
        - ◊ Figyelem, **nem a levél elküldésének dátuma**!
        - ◊ Ennek **kell id?zóna** információt is tartalmaznia, még ha UTC+0 is
        - ◊ a python `dateutil.parser.parse` függvényének fel kell tudnia olvasni
    4. további sorai a levél teste nyers szövegként

pickle

A python függvény típusú beküldéseknek a teszt fájljai bináris pickle fájlok. Ezeket kicsit körülményes szerkeszteni, ezért erre van egy segéd script.

```
python3 makepickle.py -h
```

Ez a parancssorban kapott argumentumokat pickle-özi.

Kapcsolók

- A kimenet (`-o --output`)
    - ♦ lehet stdout (ha üresen hagyjuk)
    - ♦ vagy egy fájlnév
    - ♦ vagy egy mappa, ekkor ebbe a mappába egy `i[0-9]+.pkl` nev? fájl lesz, olyan sorszámmal, ami még nincsen. Ez az új teszt bemenetekhez javasolt!
- lista vagy egy elem (`-l --list`)
    - ♦ Ha ez a kapcsoló nincsen bekapcsolva, akkor egyszer?en az els? parancssori argumentum lesz kimentve.
    - ♦ Ha be van kapcsolva, akkor az összes argumentum, mint lista lesz kimentve. A teszt bemenetekhez javasolt!