Python (Bosch)
1. fordulat

Ismertető a feladathoz

```
# kitchen.py

print('This module contains classes related to kitchen items')

class Fridge:
    def __init__(self, default_temperature):
        self.default_temperature = default_temperature
        self.food = []
        print('Fridge is ready')

class SmartScale:
    def __init__(self):
        self.measurement_history = []
        print('Scale is ready')

    def save_measurement(self, weight):
        self.measurement_history.append(weight)


print('Appliances loaded')

class KitchenHub:
    def __init__(self, appliances):
        self.appliances = appliances


    def show_appliances(self):
        pass

print('This is the end of the kitchen module')
```

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 10 perc került rögzítésre mint megoldáshoz felhasználni idő.

1. feladat 0 / 3 pont

Mit olvashatunk a kimeneten, ha a modulból a SmartScale osztályt importáljuk?

```
from kitchen import SmartScale
```

- [ ] This module contains classes related to kitchen items
  Scale is ready
  Appliances loaded
  This is the end of the kitchen module
- [ ] This module contains classes related to kitchen items
  Appliances loaded
  This is the end of the kitchen module
  Scale is ready
- [x] This module contains classes related to kitchen items
  Appliances loaded
  This is the end of the kitchen module
- [ ] This module contains classes related to kitchen items
  Scale is ready
- [ ] Scale is ready

Magyarázat a megoldáshoz

Importáláskor a python a teljes file-t végigfuttassa.

2. feladat 0 / 1 pont

A konyha után Jánosa a fürdőszobai mérlegét is szeretné hozzáadni a rendszerhez, ennek a kódja a bathroom.py file-ban a következőképpen néz ki:

```
class SmartScale:
    def __init__(self, height, age, gender):
        self.measurement_history = []
        self.height = height
        self.age = age
        self.gender = gender
        print('Scale is ready')
```

Mi lesz a kimenet, ha a fő file implementációja az alábbi?

```
from kitchen import SmartScale
from bathroom import SmartScale

if __name__ == '__main__':
    kitchen_scale = SmartScale()

    kitchen_scale.save_measurement(12.5)
```

- [ ] AttributeError: 'SmartScale' object has no attribute 'save_measurement'
- [ ] NotImplementedError: 'SmartScale' object has no function 'save_measurement'
- [ ] Scale is ready
- [x] TypeError: __init__() missing 3 required positional arguments: 'height', 'age', and 'gender'
- [ ] A programnak nincs kimenete
- [ ] TypeError: __init__() missing 4 required positional arguments: 'self', 'height', 'age', and 'gender'

Magyarázat a megoldáshoz

Az importálások fentről lefelé egymás után történnek, így a SmartScale osztály bathroom file-ból történő importálása felülírja a kitchen file-belit.

3. feladat 0 / 1 pont

A KitchenHub-hoz csatlakoztatott kijelzőn szeretnék megjeleníteni az elérhető eszközöket. Ehhez implementáljuk a show_appliances metódust.

Az alábbiak közül melyik megoldások nem alkalmasak erre?

- [ ]
```
def show_appliances(appliances):
    for appliance in appliances:
        print(appliance["name"])
```
- [x]
```
def show_appliances(appliances):
    idx = 0
    while(idx < len(appliances)):
        print(appliances[idx]["name"])
        idx+=1
```
- [x]
```
def show_appliances(appliances):
    for idx in range(len(appliances)):
        print(appliances[idx]["name"])
```
- [ ]
```
def show_appliances(appliances):
    idx = 0
    while(idx < len(appliances)):
        print(appliances[idx]["name"])
        idx+=1
```
- [x]
```
def show_appliances(appliances):
    idx = 0
    while(idx < len(appliances)):
        print(appliances[idx]["name"])
        idx++
```
- [x]
```
def show_appliances(appliances):
    idx = 0
    do:
        print(appliances[idx]["name"])
        idx++1
        while(idx < len(appliances))
```

Magyarázat a megoldáshoz

```
def show_appliances(appliances):
    idx = 0
    while(idx < len(appliances)):
        print(appliances[idx]["name"])
```

: a megoldásban hiányzik az idx változó növelése, így egy végtelen ciklusba kerülünk, ami csak az első eszközt írja ki.

```
def show_appliances(appliances):
    idx = 0
    while(idx < len(appliances)):
        print(appliances[idx]["name"])
        idx++
```

: megoldásban egy olyan syntax sugar van használva idx növelésére, ami az alap pythonban nem elérhető

```
def show_appliances(appliances):
    idx = 0
    do:
        print(appliances[idx]["name"])
        idx++1
        while(idx < len(appliances))
```

: megoldásban pedig do-while ciklus van, ami nem létezik a pythonban.

```
def show_appliances(appliances):
    for idx in range(len(appliances)):
        print(appliances[idx]["name"]):
```

Az "appliance" paraméter nem helyes.

4. feladat 0 / 1 pont

Következő lépésben János egy okostükröt szerel fel a fürdőszobába, ami reggel a legújabb hírekkel készíti:

```
class SmartMirror:
    def __init__(self):
        self.all_news = []

    def get_short_info(self):
        return [news["headline"] for news in self.all_news if "Sport" in
    def set_news(self, news):
        self.all_news = news
```

Tesztelésként a következő kódrészletet futtatja:

```
news = [
{"headline": "Az elnök szombaton sportolt", "content": "A ciprusi elnök
{"headline": "Betiltják a hajszelét", "content": "Egyes érdekvédelmek ne
{"headline": "Sportnak szánt kutyákkal sétálni", "content": "A mackóbu
{"headline": "Sport hírek: győzött a magyar csapat", "content": "A magyc
]

bathroom_mirror = SmartMirror()
bathroom_mirror.set_news(news)

for news in bathroom_mirror.get_short_info():
    print(news)
```

Milyen híreket fog olvasni?

- [ ] Az elnök szombaton sportolt
  Sportnak szánt kutyákkal sétálni
  Sport hírek: győzött a magyar csapat
- [ ] Az elnök szombaton sportolt
  Betiltják a hajszelét
  Sportnak szánt kutyákkal sétálni
  Sport hírek: győzött a magyar csapat
- [x] Sportnak szánt kutyákkal sétálni
  Sport hírek: győzött a magyar csapat
- [ ] Sport hírek: győzött a magyar csapat

Magyarázat a megoldáshoz

A substring keresés case sensitive, és csak a headline-ok közt keresünk.

5. feladat 0 / 1 pont

A könnyebb kezelhetőség miatt az egyes hírek tárolására létrehoz egy osztályt:

```
class News:
    def __init__(self, headline, content):
        self.headline = headline
        self.content = content
```

Hogyan módosítsuk a SmartMirror osztály set_news függvényét, hogy News objektumok kerüljenek az all_news listába?

Tekintettül, a hírportál API-ján keresztül a következő adat érkezik:

```
news = [
{"headline": "Az elnök szombaton sportolt", "content": "A ciprusi elnök
{"headline": "Betiltják a hajszelét", "content": "Egyes érdekvédelmek ne
{"headline": "Sportnak szánt kutyákkal sétálni", "content": "A mackóbu
{"headline": "Sport hírek: győzött a magyar csapat", "content": "A magyc
]
```

- [ ] A jelenlegi megoldás helyes
- [x]
```
def set_news(self, news):
    for item in news:
        self.all_news.append(News({item["headline"], item["content"]}))
```
- [ ]
```
def set_news(self, news):
    for item in news:
        self.all_news.append(News(item.headline, item.content))
```
- [ ]
```
def set_news(self, news):
    for item in news:
        self.all_news.append(News(item))
```

Magyarázat a megoldáshoz

Az egyes hírek az eredeti adatstruktúrában dictionary-ként vannak definiálva. A dictionary attribútumaira [] -el tudunk hivatkozni.

Vissza a kategóriákhoz

Python (Bosch)
2. forduló

Ismertető a feladathoz

A KitchenHub-al elért sikeren felbuzdulva János egy, az egész lakásra kiterjedő hub kidolgozásán fáradozik.

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 10 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 2 pont

A MainHub feladatai közé fog tartozni a wifi hálózatra kapcsolódó eszközök kiilistázása. Ehhez János a lenti függvényt írja.

A program futtatásakor hibaüzenetet kap. Mi lesz az?

```
import random

def list_available_devices(devices, black_list):
    available = []
    for device in devices:
        try:
            if device["id"] in black_list:
                print("Dangerous device detected")
            else:
                available.append(device)
        except AttributeError:
            print("Registering new device")
            device["id"] = random.randint(1,100000)
            available.append(device)

    return available


found_devices = [{"id": 1, "name": "Smart lamp"}, {"id": 2, "name": "Sma
black_list = [4]

available_devices = list_available_devices(found_devices, black_list)
```

```
  if device["id"] in black_list:
                    ^
SyntaxError: invalid syntax
```

- [x] KeyError: 'id'
- [ ] ValueError: 'id'
- [ ] AttributeError: 'list' object has no attribute 'append'

Magyarázat a megoldáshoz

Az if device["id"] in black_list: utasítással megpróbáljuk lekérni a dictionary egy olyan elemét, ami nem biztos, hogy létezik. A dictionary nemlétező kulcsára a pythonban külön error típus van: a KeyError

2. feladat 0 / 1 pont

A megtalált eszközöket ezután szeretné regisztrálni. Erre a következő függvényt írja:

```
def register_devices(devices):
    def setup_device(device_name, _id):
        name = device_name
        _id_no = _id

    for device in devices:
        setup_device(device["name"], device["id"])
        print("Device setup done: {device}".format(device-name))

register_devices(available_devices)
```

```
Sajnos János ma nincs jó formában, ez a kód sem működik!
Mint minden igazi szakember, János is az internet mélyéhez fordul segítségül.
Melyik megoldási javaslatot fogadja el?
```

- [x]
```
def register_devices(devices):
    def setup_device(device_name, _id):
        nonlocal name
        name = device_name
        _id_no = _id

    name = None
    for device in devices:
        setup_device(device["name"], device["id"])
        print("Device setup done: {device}".format(device-name))
```

- [ ]
```
def register_devices(devices):
    def setup_device(device_name, _id):
        global name
        name = device_name
        _id_no = _id

    name = None
    for device in devices:
        setup_device(device["name"], device["id"])
        print("Device setup done: {device}".format(device-name))
```

- [ ]
```
def register_devices(devices):
    def setup_device(device_name, _id):
        name = device_name
        _id_no = _id

    name = None
    for device in devices:
        setup_device(device["name"], device["id"])
        print("Device setup done: {device}".format(device-name))
```

- [ ]
```
def register_devices(devices):
    def setup_device(device_name, _id):
        nonlocal name = device_name
        _id_no = _id

    name = None
    for device in devices:
        setup_device(device["name"], device["id"])
        print("Device setup done: {device}".format(device-name))
```

- [ ]
```
def register_devices(devices):
    def setup_device(device_name, _id):
        nonlocal name = device_name
        _id_no = _id

    for device in devices:
        setup_device(device["name"], device["id"])
        print("Device setup done: {device}".format(device-name))
```

Magyarázat a megoldáshoz

Ahhoz, hogy egy függvény lokális változója a függvényen kívül is elérhető legyen, a nonlocal keyword-öt kell használnunk. A nonlocal helyes működéséhez az egyel kiljebbi scope-ban definiálnunk kell ugyanazon a néven egy változót.

3. feladat 0 / 1 pont

Amíg a kérdésre kapott válaszok közt böngészve János elgondolkozik a python keyword-jein. Korábbi, C-ben szerzett tapasztalatából kiindulva összehasonlítja, hogy mely keyword-ök léteznek a python-ban is, és melyek nem.

Az alábbiak közül melyik nem python keyword?

- [ ] global
- [x] goto
- [ ] as
- [ ] del
- [x] list
- [ ] await

Magyarázat a megoldáshoz

A goto főleg a C nyelvre jellemző. A list típus, nem keyword.

4. feladat 0 / 1 pont

Ezután felfedezi, hogy hibakezelésre sem csak az except használható. Melyik keyword nem használható hibakezelésre az alábbiak közül?

- [ ] try
- [x] elif
- [ ] finally
- [ ] assert
- [ ] else

Magyarázat a megoldáshoz

Az else keyword esetenként megjelenhet az except ág után. Ez akkor fut le, ha nem futottunk bele az except ágba, az esetek többségében nem különbözik attól, mintha simán folytatnánk a kódirást.

Vissza a kategóriáimhoz

Python (Bosch)
3. forduló

Ismertető a feladathoz

János figyelme a hasára terelődik... és onnan a hűtőjére. Vajon van itthon profiterol? Akár a telefonja is megmondhatná!

A korábbi Fridge osztályt kiegészíti egy úgy adattaggal:

```
class Fridge:
    def __init__(self, default_temperature):
        fridge_contents = [1,2,2,2,5,7,7,9,9,9,9,8,3,3,3]
        self.default_temperature = default_temperature
        self.food = []
        print('Fridge is ready')
```

A fridge_contents a hűtőben található élelmiszerek azonosítóját tartalmazza. Az élelmiszerek mennyiségét az egyes azonosítók darabszáma fejezi ki.

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 15 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 2 pont

Az ételek azonosítóját János a következőként tárolja:

```
food_map = {
    "Alma": 1,
    "Csirkemell": 2,
    "Túró rudi": 3,
    "Tojás": 4,
    "Paradicsom": 5,
    "Profiterol": 6,
    "Hagyma": 7,
    "Brie sajt": 8,
    "Szalámi": 9
}
```

Az alábbiak közül melyik lambda adja meg, hogy van-e a hűtőben profiterol?

- ```
  check_for_food = lambda x: True if x in food_map else False
  ```
- ```
  check_for_food = lambda x: False if x in food_map else True
  ```
- ```
  check_for_food = lambda x: True if food_map[x] in fridge_contents els
  ```
- ```
  check_for_food = lambda x: False if food_map[x] in fridge_contents el
  ```
- ```
  check_for_food = lambda x: True if x in fridge_contents else False
  ```
- ```
  check_for_food = lambda x: False if x in fridge_contents else True
  ```
- ```
  check_for_food = lambda x: True if fridge_contents[x] in food_map els
  ```
- ```
  check_for_food = lambda x: False if fridge_contents[x] in food_map el
  ```

Magyarázat a megoldáshoz

Ha a "Profiterol" stringre keresünk rá, a hozzátartozó azonosítót a food_map objektumból kapjuk meg: food_map[x]. Ezt az azonosítót a fridge_contents listában keressük.

2. feladat 0 / 1 pont

János szereti, ha bőven van otthon étel. Ha valamiből kettőnél kevesebb van, szeretne értesítést kapni. Melyik kódrészletek listázzák ki helyesen a szükséges élelmiszerek azonosítóját?

- [x]
  ```
  need_to_buy = []
  what_to_buy = lambda x: need_to_buy.append(x) if fridge_contents.coun
  for item in fridge_contents:
      what_to_buy(item)
  print(need_to_buy)
  ```
- [x]
  ```
  need_to_buy = []
  what_to_buy = lambda x: need_to_buy.append(x) if fridge_contents.coun
  for item in fridge_contents:
      what_to_buy(item)
  print(need_to_buy)
  ```
- [ ]
  ```
  need_to_buy = []
  what_to_buy = lambda x: x if fridge_contents.count(x) < 2 else 0
  for item in fridge_contents:
      what_to_buy(item)
  print(need_to_buy)
  ```
- [ ]
  ```
  what_to_buy = lambda x: x if fridge_contents.count(x) < 2 else None
  need_to_buy = [what_to_buy(x) for x in fridge_contents]
  print(need_to_buy)
  ```
- [ ]
  ```
  need_to_buy = []
  what_to_buy = lambda x: x if need_to_buy.count(x) < 2 else 0
  need_to_buy = [what_to_buy(x) for x in fridge_contents]
  print(need_to_buy)
  ```
- [ ]
  ```
  need_to_buy = []
  what_to_buy = lambda x: need_to_buy.append(x) if need_to_buy.count(x)
  for item in fridge_contents:
      what_to_buy(item)
  print(need_to_buy)
  ```

Magyarázat a megoldáshoz

A need_to_buy = []
what_to_buy = lambda x: x if fridge_contents.count(x) < 2 else 0
for item in fridge_contents:
  what_to_buy(item)
print(need_to_buy)
megoldásnál a logika helyes, de nem adjuk hozzá a need_to_buy listához az elemeket.

A többi megoldásnál nem csak az elemeket adjuk hozzá a listához.

3. feladat 0 / 1 pont

János egyre jobban kedveli a pythont. Úgy dönt, a lambdákat is jobban átnézi. Az első dolog ami felmerül benne: Mi lesz a what_to_buy objektum típusa?

- object
- lambda
- function
- list
- map
- method

Magyarázat a megoldáshoz

A lambda sima függvénynek számít, anonymous function-ként is szokás rá hivatkozni.

4. feladat 0 / 1 pont

A következő dolog amin elgondolkozik: ha lambda egy függvény, mi a különbség az alábbi két implementáció közt?

```
l = lambda x: x*2

def multiplier(x):
    return x*2
```

- l típusa function míg multiplier típusa method
- Bytekód szinten található különbség: l először betölti a két értéket, majd végrehajtja a szorzást, míg multiplier először betölti x-et, majd a szorzás műveletet, majd a 2-es konstanst
- Nincs különség, a két megoldás minden szempontból azonos
- divider string típusokra is működik, míg l nem

Magyarázat a megoldáshoz

Az előző feladathoz hasonlóan: a lambda ugyanolyan function, mint a "sima" def-el íródott verzió.

Vissza a kategóriáimhoz

Python (Bosch)
4. forduló

Ismertető a feladathoz

Május felé János figyelme a kertje felé fordul: ahogy melegszik az idő, egyre inkább szárad a fű. Itt az ideje a kertet is felokosítani. A locsolók vezérlésére, és az adatok házból történő monitorozására a következő kódot írja:

```
import pickle

class Sprinkler:
    def __init__(self, sprinkler_name):
        self.sprinker_name = sprinkler_name
        self.is_registered = False

    def setup_sprinkler(self):
        self.is_registered = True
        return "Setup done"

    def get_data(self):
        return self.sprinkler_name


class SprinklerData(dict):
    def __init__(self, *sprinkler_names):
        self.sprinklers = [Sprinkler(sprinkler_name) for sprinkler_name

    def __getattr__(self, key):
        return self[key]

    def data_getter(self):
        pass

    def data_copyer(self):
        pass

data_handler = SprinklerData("Sprinkler1", "Sprinkler2")

data_handler.data_getter()
```

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 10 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 2 pont

A SprinklerData data_getter függvényére a következőt találja ki:

```
def data_getter(self):
        data_getters = []
        for sprinkler in self.sprinklers:
            def fu():
                return self.sprinkler.get_data()
            data_getters.append(fu)

        with open("test", "wb") as test_file:
            pickle.dump(data_getters, test_file)
```

Mi lesz a kimenete a kódnak ezzel az implementációval?

- Sprinkler1 Sprinkler2
- AttributeError: Can't pickle local object 'SprinklerData.data_getter.<locals>.fu'
- ValueError: Can't pickle local object 'SprinklerData.data_getter.<locals>.fu'
- AttributeError: Can't pickle local object 'SprinklerData.data_getter

Magyarázat a megoldáshoz

A pickle szerializáláskor csak a metódusok nevét menti el, és deszerializáláskor újra beimportál mindent. Viszont beimportálni csak a top-level funkciókat lehet, ami itt nem valósul meg.

2. feladat 0 / 1 pont

Következő lépésben a data_copyer függvényt írja meg. Melyik implementáció nem fog működni?

```
def data_copyer(self):
    return self.sprinklers[:]
```

```
def data_copyer(self):
    return self.sprinklers
```

```
def data_copyer(self):
    from copy import deepcopy
    return deepcopy(self.sprinklers)
```

```
def data_copyer(self):
    from copy import shallowcopy
    return shallowcopy(self.sprinklers)
```

Magyarázat a megoldáshoz

Nem létezik külön shallowcopy metódus, mert minden másolásra szolgáló módszer, ami nem deepcopy, az shallow copy-t eredményez.

3. feladat 0 / 1 pont

Az alábbiak közül melyik nem szerializálható a pickle package segítségével?

- [ ] class
- [x] defaultdict
- [ ] komplex számok
- [x] lambda függvények

Magyarázat a megoldáshoz

A pickle csak olyan függvényeket tud szerializálni, amelyek "importálhatóak". A lambda kifejezésekre ez nem igaz.

Az alap defaultdict valóban szerializálható.

4. feladat 0 / 1 pont

Az alábbiak közül melyik segítségével lehet lambda függvényeket szerializálni?

- pumpkin
- marshal
- dill
- tomato

Magyarázat a megoldáshoz

A dill a pickle funkcionalitását egészíti ki azzal, hogy a konkrét objektumon kívül az interpreter state-jét is szerializálja.

Vissza a kategóriáimhoz

Python (Bosch)
5. forduló

Ismertető a feladathoz

János napi sétáját általában a helyi egyetem füvészkertjében végzi. Egyik nap arra lesz figyelmes, hogy egy új beruházás miatt az informatika tanszék kiárusítja a régi eszközparkot. János viszonylag olcsón vesz egy szerverszámítógépet, de otthon észreveszi, hogy véletlenül egy 12000 magos szuperszámítógépet adtak el neki.

Új dönt, a szuperszámítógépet az öntözőrendszeren próbálja ki először. A már korábban megírt Sprinkler osztályt a következő módon egészíti ki:

```
class Sprinkler:
    def __init__(self, sprinkler_name:str, water_tank:Queue):
        self.sprinker_name = sprinkler_name
        self.is_registered = False
        self.water_tank = water_tank

    def __iter__(self):
        return self

    def __next__(self):
        if not self.water_tank.empty():
            return self.water_tank.get()
        else:
            raise StopIteration

    def setup_sprinkler(self):
        self.is_registered = True
        return "Setup done"

    def get_data(self):
        return self.sprinkler_name


class Droplet:
    def __init__(self, quantity):
        self.quantity = quantity

    def __add__(self, other):
        Droplet(self.quantity + other.quantity)

    def __repr__(self):
        return 'splash ' * self.quantity
```

A teszteléshez még további segédfüggvényeket is készít:

```
def do_wait(n):
    time.sleep(n)


def provide_water(water_tank:Queue):
    while True:
        if not water_tank.full():
            water_tank.put(Droplet(1))
            do_wait(1)
        else:
            break


def leak_water(water_tank:Queue):
    while True:
        if not water_tank.empty():
            water_tank.get()
            do_wait(2)
        else:
            break
```

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 15 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 5 pont

Az elkészült kódot először a vízfeltöltéssel szeretné kipróbálni:

```
def main():
    t0 = time.time()
    water_tank = Queue(8)
    provider = Thread(target-provide_water, args-(water_tank,))
    provider.start()
    tap = Sprinkler("Sprinkler", water_tank)

    do_wait(3)
    for droplet in tap:
        print(droplet, end-'')

    print(time.time() - t0)
    provider.join()

main()
```

Mi lesz a main() függvény hívásának kimenete?

- <__main__.Droplet object at ...> <__main__.Droplet object at ...> <__main__.Droplet object at ...>
- Nem ír ki semmit
- 3
- splash splash splash 3
- splash splash splash

Magyarázat a megoldáshoz

A water_tank Queue háromszor tud feltöltődni plusz egy elemmel, amíg a for ciklus azt kimeríti. A Droplet osztály reprezentációja: splash

2. feladat 0 / 5 pont

Ezután a szivárgást is teszteli:

```
def main():
    t0 = time.time()
    water_tank = Queue(8)
    provider = Process(target-provide_water, args-(water_tank,))
    leak = Process(target-leak_water, args-(water_tank,))
    provider.start()
    leak.start()
    tap = Sprinkler("Sprinkler1", water_tank)

    do_wait(3)
    for droplet in tap:
        print(droplet, end-'')

    print(time.time() - t0)
    provider.join()
    leak.join()

main()
```

Mi lesz a main() függvény hívásának kimenete?

- splash splash
- <__main__.Droplet object at ...> <__main__.Droplet object at ...>
- [x] 3
- [x] splash 3
- Nem ír ki semmit

Magyarázat a megoldáshoz

A leak_water 3 mp alatt kétszer jut el oda, hogy egy cseppet ki vegyen a Queue-ból.

3. feladat 0 / 2 pont

Tegyük fel, hogy a time.sleep() helyében egy számítás igényes feladat áll.
Hogyan változik meg az egész script lefutásához (átlagosan) szükséges idő?

- Több mint kétszer annyi lesz
- Nem változik
- Kevesebb lesz
- Kicsit több lesz

Magyarázat a megoldáshoz

A Python Global Interpreter Lock csak virtuálisan engedi a threadeket hogy egyszerre fussanak, a valóság az, hogy felváltva futtnak, így a hosszú számításokat nem tudja gyorsabban elvégezni hiába van sok mag.

4. feladat 0 / 3 pont

Hogyan változik az előző esethez képest a futás idő, ha Thread helyett multiprocessing.Process() -eket használunk?

- Futás során hibába ütközünk
- Kevesebb lesz
- Kicsit több lesz
- Nem változik

Magyarázat a megoldáshoz

A Process-ek indításával külön processeket indítunk a számítógépen, amiket már nem két a GIL, így a gép ki tudja használni a több processzorát.

5. feladat 0 / 3 pont

Hogyan változik a futás idő ha a Sprinkler osztály __next__() metódusát a következőre cseréljük?

```
def __next__(self):
    return self.water_tank.get()
```

- [x] Nem változik
- Hibába ütközik
- [x] Végtelen lesz
- Jelentősen lecsökken
- Egy kicsit megnő

Magyarázat a megoldáshoz

Egy iterable objectumon való iterálás addig tart amíg a __next__() metódus nem dob StopIteration hibát, ha ez soha nem történik meg, az iteráció sem ér véget

Vissza a kategóriákhoz

Python (Bosch)
6. forduló

Ismertető a feladathoz

János robotokkal kísérletezik: Először egy robotporszívót vesz, majd egy robotfelmosót, aztán megtalálja az igazit: Robi mindkettőt tudja. János nem szereti a dupla munkát: úgy dönt megtanítja a korábbi eszközökhöz készített implementációt, és abból örökíti Robi osztályát:

```
class RobotVacuum:
    def __init__(self, battery_capacity):
        self.battery_capacity = battery_capacity
        print("Robot Vacuum setup done")
        super().__init__()

class RobotMop:
    def __init__(self, battery_capacity, water_capacity):
        self.battery_capacity = battery_capacity
        self.water_capacity = water_capacity
        print("RobotMop init")

class MagicCleaner(RobotMop, RobotVacuum):
    def __init__(self, battery_capacity, water_capacity):
        super(MagicCleaner, self).__init__(battery_capacity, water_capac
        print("MagicCleaner ready to help!")

robi = MagicCleaner(1800, 2)
```

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 20 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 9 pont

Mit olvas a kimeneten, amikor megpróbálja Robit példányosítani?

- ( ) TypeError: Cannot create a consistent method resolution order (MRO) for bases RobotMop, RobotVacuum
- ( ) RobotMop init
Robot Vacuum setup done
MagicCleaner ready to help!
- ( ) MagicCleaner ready to help!
Robot Vacuum setup done
RobotMop init
- ( ) MagicCleaner ready to help!
- (•) RobotMop init
MagicCleaner ready to help!
- ( ) RobotMop init
Robot Vacuum setup done
MagicCleaner ready to help!

Magyarázat a megoldáshoz

A MagicCleaner osztályban található super() hívás csak az első ős __init__ függvényét hívja meg.

2. feladat 0 / 1 pont

Mi lesz a Method Resolution Order a MagicCleaner class esetén?

- (•) MagicCleaner, RobotMop, RobotVacuum, object
- ( ) RobotMop, RobotVacuum, MagicCleaner, object
- ( ) MagicCleaner, RobotMop, RobotVacuum
- ( ) RobotMop, RobotVacuum, object
- ( ) MagicCleaner, object, RobotMop, RobotVacuum
- ( ) object, RobotMop, RobotVacuum, MagicCleaner
- ( ) Nincs konzisztens method resolution order

Magyarázat a megoldáshoz

Példányosításkor először a példányosítandó osztály hívódik meg, majd balról jobbra az ősosztályok, majd azok ősosztályai, míg végül eljutunk minden őséhez: az object classhoz

3. feladat 0 / 1 pont

Egy fórumon Jánosnak javasolják, hogy javítson a SmartScale osztályán:

```
class SmartScale:
    def __init__(self, height, age, gender):
        self.height = height
        self.age = age
        self.gender = gender

class SmartScale2:
    __slots__ = ["height", "age", "gender"]

    def __init__(self, height, age, gender):
        self.height = height
        self.age = age
        self.gender = gender
```

A különbség megértéséhez János ír egy segédfüggvényt, és kiírja az eredményt:

```
def display_stats(obj):
    for stat in obj.__dict__.items():
        print(stat)

smart_scale = SmartScale("173", "22", "male")
display_stats(smart_scale)

smart_scale2 = SmartScale2("173", "22", "male")

display_stats(smart_scale2)
```

Mit fog látni János?

- ( ) ('height', '173')
('age', '22')
('gender', 'male')
('height', '173')
('age', '22')
('gender', 'male')
- ( ) {'height': '173', 'age': '22', 'gender': 'male'}
AttributeError: 'SmartScale2' object has no attribute '__dict__'
- ( ) AttributeError: 'SmartScale' object has no attribute '__dict__'
AttributeError: 'SmartScale2' object has no attribute '__dict__'
- ( ) AttributeError: 'SmartScale' object has no attribute '__dict__'
('height', '173')
('age', '22')
('gender', 'male')
- (•) ('height', '173')
('age', '22')
('gender', 'male')
AttributeError: 'SmartScale2' object has no attribute '__dict__'

Magyarázat a megoldáshoz

A __slots__ használata azzal gyorsítja a python objektumokat, hogy nem hoz létre __dict__ objektumot az attribútumokból.

4. feladat 0 / 1 pont

János tovább kísérletezik. Mi történik, ha egy __slots__-t használó osztályból örököl?

```
class EnhancedScale(SmartScale2):
    def __init__(self, height, age, gender, weight_goal):
        super().__init__(height, age, gender)
        self.weight_goal = weight_goal

enhanced_scale = EnhancedScale(173, 22, "male", 80)
```

Mit látunk a kimeneten, ha kiprintéljük enhanced_scale.__dict__ hívást?

- ( ) AttributeError: 'EnhancedScale' object has no attribute '__dict__'
- (•) {'weight_goal': 80}
- ( ) AttributeError: 'SmartScale2' object has no attribute '__dict__'
- ( ) {'height': 173, 'age': 22, 'gender': 'male', 'weight_goal': 80}
- ( ) {'height': 173, 'age': 22, 'gender': 'male'}

Magyarázat a megoldáshoz

Ha egy osztály nem definiálja __slots__-t, generálódik dictionary az attribútumaiból. De az ősosztálya ez nem igaz, így a végső dictionary-be csak a child osztály attribútumai kerülnek be.

5. feladat 0 / 1 pont

Jánosnak megtetszik a __slots__ mechanikája. Úgy dönt refaktorálja a robotjait:

```
class RobotMop:
    __slots__ = ['water_copacity']

    def __init__(self):
        print("RobotMop init")
        super().__init__()

class RobotVacuum:
    __slots__ = ["battery_copacity"]

    def __init__(self):
        print("RobotVacuum init")

class MagicCleaner(RobotMop, RobotVacuum):
    def __init__(self):
        print("Magic cleaner ready!")
        super().__init__()

robi = MagicCleaner()
```

Mit lát a kimeneten, amikor megpróbálja Robit példányosítani?

- ( ) Magic cleaner ready!
- ( ) Magic cleaner ready!
RobotMop init
RobotVacuum init
- (•) TypeError: multiple bases have instance lay-out conflict
- ( ) Magic cleaner ready!
RobotVacuum init
RobotMop init
- ( ) Magic cleaner ready!
RobotMop init
- ( ) TypeError: Cannot create a consistent method resolution order (MRO) for bases RobotMop, RobotVacuum

Magyarázat a megoldáshoz

A child class örökli az ősök slot-jait. Ha egy class-ban vannak slot-ok, nem lehet futásidőben új attribútumot hozzáadni. Mivel a child class nem definiál egy slot-ot sem, nem lehet eldönteni, hogy mely slotokra van szükség.

Vissza a kategóriákhoz

Python (Bosch)
7. forduló

Ismertető a feladathoz

János megunja, hogy minden alkalommal külön kell implementálnia az egyes eszközök üzenetküldő funkcióit. Ismét az internethez fordul segítségért. Egy fórumon egy érdekes megoldást javasolnak neki, amit be is szúr a már létező Fridge osztálya alá:

```
class Fridge:
    def __init__(self, default_temperature):
        print("Initializing fridge...")
        fridge_contents = [1,2,2,2,5,7,7,9,9,9,9,8,3,3,3]
        self.default_temperature = default_temperature
        self.food = []
        self.send_message("Ready to cool!")

    def send_message(self, msg):
        print(msg)

class Messenger(type):
    def __new__(cls, name, bases, dct):
        print("Now setting up sender...")
        obj = super().__new__(cls, name, bases, dct)
        obj.send_message = send_message
        return obj

    def send_message(obj, msg):
        print("{appliance_name} says: {message}".format(appliance_name=c

class Appliance(metaclass=Messenger):
    def __init__(self):
        print("Initializing Appliance...")
        self.send_message("Ready to work!")

    def __repr__(self):
        return "Appliance"
```

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 20 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 5 pont

Hogy kipróbálja az új üzenetküldő funkciót, amit az Appliance osztály tesz lehetővé, a kávéfőzőjét is "felokosítja". Az ehhez tartozó osztály a következő:

```
class CoffeeMaker(Appliance):
    def __init__(self):
        print("Initializing coffee maker...")
        self.send_message("Ready to brew!")

    def send_message(self, msg):
        print(msg)
```

Mit lát János a kimeneten példányosításkor?

- [ ] Now setting up sender...
  Initializing coffee maker...
  CoffeeMaker says: Ready to brew!
- [ ] Initializing coffee maker...
  Now setting up sender...
  Appliance says: Ready to brew!
- [x] Now setting up sender...
  Initializing coffee maker...
  Appliance says: Ready to brew!
- [ ] Initializing coffee maker...
  Now setting up sender...
  CoffeeMaker says: Ready to brew!
- [x] Initializing coffee maker...
  Appliance says: Ready to brew!
- [ ] Initializing coffee maker...
  CoffeeMaker says: Ready to brew!

Magyarázat a megoldáshoz

Egy objektum példányosításakor python-ban először az osztály __new__() metódusa hívódik meg. Ez alapesetben visszaadja az új példányt, majd ennek az új példánynak az __init__() metódusa fut le. Mivel CoffeeMaker osztály nem definiál __new__ függvényt, az ősének definíciója kerül felhasználásra.
A send_message függvény szintén azért nem CoffeeMaker-ként hivatkozik a példányunkra, mert a CoffeeMaker osztály nem definiálja felül az Appliance osztály __repr__ függvényét.

2. feladat 0 / 1 pont

János kíváncsi lesz: mi az egyes osztályok típusa?

```
print(type(Messenger))
print(type(Fridge))
print(type(CoffeeMaker))
print(type(Appliance))
```

- ( ) Messenger: <class '__main__.Messenger'>
  Fridge: <class '__main__.Fridge'>
  CoffeeMaker: <class '__main__.CoffeeMaker'>
  Appliance: <class '__main__.Appliance'>
- ( ) Messenger: <class 'type'>
  Fridge: <class 'type'>
  CoffeeMaker: <class 'type'>
  Appliance: <class 'type'>
- ( ) Messenger: <class 'type'>
  Fridge: <class 'type'>
  CoffeeMaker: <class '__main__.CoffeeMaker'>
  Appliance: <class '__main__.Appliance'>
- (•) Messenger: <class 'type'>
  Fridge: <class 'type'>
  CoffeeMaker: <class '__main__.Messenger'>
  Appliance: <class '__main__.Messenger'>
- ( ) Messenger: <class 'type'>
  Fridge: <class 'type'>
  CoffeeMaker: <class '__main__.Appliance'>
  Appliance: <class '__main__.Messenger'>

Magyarázat a megoldáshoz

A pythonban minden objektumnak számít, az osztályokat is beleértve. Ezért értelmezhető az osztályok típusa. A python 3-ban egyéb definíció nélkül minden osztály a type típusból öröklődik, így típusuk type. Ha viszont egy osztálynak metaclass-t definiálunk, felülírjuk a típusát, mert nem közvetlenül a type-ból öröklődik.

3. feladat 0 / 1 pont

János ezután beszerez egy új termosztátot, amit a konyhába tervez beszerelni. A gyártó biztosít egy python package-t a vezérléshez, ez egy absztrakt osztályt tartalmaz, melynek neve SchobThermostat.
János nem híve a dokumentáció elolvasásának, így úgy dönt, szimplán belevág, és implementálja a saját osztályát:

```
class Thermostat(SchobThermostat, Appliance):
    def __init__(self, is_on=False):
        print("Initializing thermostat...")
        self.current_temperature = 22.0
        self.get_temperature()
        self.goal_temperature = 22.0
        self.on = is_on
        print("Initialization done. Thermostat is now {on}".format(on="c

    def run(self):
        while(self.on):
            self.get_temperature()
            if (self.current_temperature < self.goal_temperature):
                for temp_increment in range(self.goal_temperature - self
                    self.current_temperature += temp_increment

    def get_temperature(self):
        print("Fetching temperature...")
        self.current_temperature = self.measure()

    def set_temperature(self, desired_temp):
        self.goal_temperature = desired_temp

thermostat = Thermostat(is_on=True)
```

Mit fog látni a kimeneten?

- ( ) Initializing thermostat...
  Fetching temperature...
  Initialization done. Thermostat is now on
- ( ) Initializing thermostat...
  Fetching temperature...
  Initialization done. Thermostat is now off
- ( ) Initializing thermostat...
  Fetching temperature...
  Initialization done. Thermostat is now True
- ( ) Initializing thermostat...
  Fetching temperature...
  Initialization done. Thermostat is now False
- (•) TypeError: metaclass conflict: the metaclass of a derived class must be a (non-strict) subclass of the metaclasses of all its bases
- ( ) TypeError: Cannot create a consistent method resolution order (MRO) for bases SchobThermostat, Appliance

Magyarázat a megoldáshoz

Egy objektumnak egy típusa lehet. Ha mindkét ős különböző metaclassal rendelkezik(ahol a metaclassok nem egymásból öröklődnek), a python nem tudja eldönteni, hogy melyik metaclass-t használja.

4. feladat 0 / 1 pont

János lemond az Appliance osztályból való öröklésről:

```
class Thermostat(SchobThermostat):
    def __init__(self, is_on=False):
        print("Initializing thermostat...")
        self.current_temperature = 22.0
        self.get_temperature()
        self.goal_temperature = 22.0
        self.on = is_on
        print("Initialization done. Thermostat is now {on}".format(on="c

    def run(self):
        while(self.on):
            self.get_temperature()
            if (self.current_temperature < self.goal_temperature):
                for temp_increment in range(self.goal_temperature - self
                    self.current_temperature += temp_increment

    def get_temperature(self):
        print("Fetching temperature...")
        self.current_temperature = self.measure()

    def set_temperature(self, desired_temp):
        self.goal_temperature = desired_temp

thermostat = Thermostat()
```

Sajnos a kód hibás. A stack trace a következő:

```
TypeError: Can't instantiate abstract class Thermostat with abstract methods
set_timer
```

Az alábbiak közül melyik implementáció nem javítja ki a hibát?

- ( )
  ```
  def set_timer(self):
      pass
  ```
- ( )
  ```
  def set_timer(self, desired_delay):
      self.desired_delay = desired_delay
  ```
- (•)
  ```
  @abc.abstractmethod
  def set_timer(self, desired_delay):
      pass
  ```
- ( )
  ```
  def set_timer(self, desired_delay):
      pass
  ```
- ( )
  ```
  def set_timer(self, desired_delay):
      super().set_timer(desired_delay)
  ```

Magyarázat a megoldáshoz

Az @abc.abstractmethod dekorátor azt jelzi, hogy függvény implementációját továbbra is egy örökölt osztályban várjuk. Egy abstract osztályból öröklődő osztályt nem példányosíthatunk anélkül, hogy az abstract osztály minden függvényét ne implementálnánk.

5. feladat 0 / 1 pont

Mi történik ha János a hiba javítása előtt megpróbál egy olyan osztályt példányosítani, ami a Thermostat osztályból öröklődik?

```
class KitchenThermostat(Thermostat):
    def __init__(self, is_on=False):
        super().__init__()

kitchen_thermostat = KitchenThermostat(is_on=False)
```

- ( ) Az osztály hiba nélkül példányosítható, mert nem közvetlenül az absztrakt osztályból származik
- ( ) Hibaüzenetet kapunk: mindegyik absztrakt osztálybeli metódust újra kell definiálnunk
- ( ) Hibaüzenetet kapunk: a thermostat osztály összes metódusát újra kell definiálnunk
- (•) Hibaüzenetet kapunk: a KitchenThermostat abstact osztály nem példányosítható a set_timer függvény nélkül
- ( ) Hibaüzenetet kapunk: a Thermostat abstact osztály nem példányosítható a set_timer függvény nélkül

Magyarázat a megoldáshoz

Mindegy milyen "messze" vagyunk öröklésben az absztakt osztálytól: az osztályunk absztraktnak számít, és minden metódust az eredeti ősosztályból implementálnunk kell

6. feladat 0 / 1 pont

Mi lesz a kimenet, ha János a KitchenThermostat set_timer függvényét az alábbi módon definiálja?

```
def set_timer(self, desired_delay):
        super().set_timer()
        print("Setting delay for {delay}...".format(delay=desired_delay)

# Mi lesz a kimenete a függvényhívásnak?
kitchen_thermostat.set_timer(25.0)
```

- ( ) Setting delay for 25.0...
- ( ) Desired delay is set
  Setting delay for 25.0...
- ( ) <bound method SchobThermostat.set_timer of <__main__.KitchenThermostat object at 0x0000029E69098FD0>>
  Setting delay for 25.0...
- ( ) <bound method KitchenThermostat.set_timer of <__main__.SchobThermostat object at 0x0000029E69098FD0>>
  Setting delay for 25.0...
- (•) Az adott kódrészletek alapján nem lehet megmondani

Magyarázat a megoldáshoz

Mivel SchobThermostat implementációja János számára nem ismert, ezért nem tudjuk, hogy van-e, és ha igen, milyen a kimenete az eredeti set_timer() függvénynek.