PYTHON IN CLOUD (ANGOL NYELVŰ)

6. forduló

CAMBRIDGE MOBILE TELEMATICS

A kategória támogatója: Cambridge Mobile Telematics (CMT)

Ismertető a feladathoz

Felhasznált idő: 00:00/35:00 **Elért pontszám:** 0/16

1. feladat 0/5 pont

Terminal

What do you see in the terminal and why?

```
def concat_with(text, items=[]):
    items.append(text)
    return items

print(sum([*concat_with(1, [2]),
           *concat_with(3),
           *concat_with(4)]))
```

Válasz

- [✓ I see 13, because of the mutable default argument]
  Ez a válasz helyes, de nem jelölted meg.
- I see 13, because the sum function can not handle the unpack operator inside a list param
- I see 13, because the unpack operator creates a duplicated iterator
- I see 10, because 1+2+3+4 == 10

Magyarázat

Python's default arguments are evaluated once when the function is defined, not each time the function is called (like it is in say, Ruby). This means that if you use a mutable default argument and mutate it, you will and have mutated that object for all future calls to the function as well.

2. feladat 0/1 pont

Scope

We have the following code snippet:

```python
scope = 42

def increase_scope_by_one():
   scope += 1
   print(f'new scope: {scope}')

def do():
   e = None

   try:
       increase_scope_by_one()
       increase_scope_by_one()

   except (ValueError, UnboundLocalError) as e:
       print('Oups')

   print(scope)
   print(e)

try:
   do()
except:
   print('No more')
```

Considering that after this code execution (python3) we would like to see the following on the standard output:

```
new scope: 43
new scope: 44
44
None
```

What bugs should be fixed in the code, to see the expected output?

Válaszok

- ☐ i.) Incorrectly used class variable
- ✓ ii.) Incorrectly used scoping rules (LEGB rule)
  Ez a válasz helyes, de nem jelölted meg.
- ☐ iii.) Name clashing with Python Standard Library modules
- ✓ iv:) Exception object accessibility beyond the scope
  Ez a válasz helyes, de nem jelölted meg.
- ☐ v.) All of them

Magyarázat

i.) no class here

ii.) scope = 42 not accessible in the increase_scope_by_one function, you would see UnboundLocalError there

iii.) no clashing here

iv.) it is a python 2 vs python 3 difference, in python 3 the exception object (e) is not accessible beyond the scope of the except block.

v.) trivial no

3. feladat 0/10 pont

Recursion 2

We have defined a function f. While we are not quite sure what it does, we are sure it's important.

For small values of x, we see that f(x):

```
print(f(1))
1
print(f(2))
3
print(f(10))
199
print(f(100))
169999
```

But for larger values of x though the code will not work as-is (you can try it yourself). We'd like to

know what f returns for the largest values of x.

```
def f(x):
    if x == 1:
        return 1
    n = f(x - 1)
    m = n % x
    return x * m + n
```

Specifically what should f(10001) return?

Válasz

A helyes válasz:

166716661665

Magyarázat

This doesn't work by default in Python because the recursion depth limit is set fairly low by default.

There are at least 3 ways to solve this, 1 of which is more of a workaround.

1. Rewrite f to be iterative. Here, the typical approach would be to use a loop to effectively build up to f(x) for the desired x by calculating each previous f(i) for i between 1 and x. An example solution is labeled below this list (same code structure as the example function maintained for clarity).
2. Memoize f, then loop calls from 1 through 10000 to fill up the memo. Going in this order ensures the call stack never goes deeper than a single recursive call because it will get a hit in the memo. An example is provided below.
3. Import sys and alter the recursion depth limit to be high enough, then call f(10001). Note that this solution won't work when other things are making the recursive function unable to work, e.g. inefficiencies, memory limits, etc.

Example solutions:

```
# Solution 1
def f(x):
    out = 1
    for i in range(2, x + 1):
        n = out                 # out is currently f(i-1)
        m = n % i
        out = i * m + n
    return out

print(f(10001))

# Solution 2
memo = {}
def f(x):
    if x == 1:
        return 1
    if x not in memo:
        n = f(x - 1)
        m = n % x
        memo[x] = x * m + n
    return memo[x]

for i in range(1, 10000):
    f(i)

print(f(10001))
```