PYTHON IN CLOUD (ANGOL NYELVŰ)

7. forduló

CAMBRIDGE MOBILE TELEMATICS

A kategória támogatója: Cambridge Mobile Telematics (CMT)

Ismertető a feladathoz

Felhasznált idő: 00:00/35:00 **Elért pontszám:** 0/25

1. feladat 0/5 pont

Cleanup

What is the proper way of calling the cleanup function? We want to ensure that the cleanup will always be called.

```
def cleanup():
    print('cleanup')

def do_the_work():
    print('do the work')

def init():
    print('init')

class MyError(Exception):
    pass

def throw_the_error():
    raise Exception()
```

Válasz

```
def main1():
    try:
        init()
        do_the_work()
    finally:
        cleanup()
```

Ez a válasz helyes, de nem jelölted meg.

```
def main2():
    try:
        init()
        do_the_work()
        cleanup()
    except:
        cleanup()
```

```
def main3():
    try:
        init()
        do_the_work()
    except:
        pass
    else:
        cleanup()
```

```
def main4():
    init()
    do_the_work()
    cleanup()
```

Magyarázat

The final block is always executed no matter what happens in the try..except block. The other examples can miss the cleanup or swallow an exception if an error happens in the init or in the do_the_work method.

2. feladat 0/10 pont

Trampolines

A colleague has provided fully-functioning functions delay and f, and has provided a skeleton of the function go they would like you to implement. A few test cases are provided when this file is executed.

```
from typing import Callable, Tuple, TypeVar, Union

T = TypeVar("T")

Delayed = Union[T, Callable[[], T]]

def delay(f, *args):
    return lambda: f(*args)

def f(z: int) -> Delayed[Tuple[int, int]]:
    if z <= 0:
        raise ValueError(f"z == {z} <= 0")

    def g(x: int, y: int, z: int) -> Delayed[Tuple[int, int]]:
        if z == 1:
            return x, y

        if z % 2 == 0:
```

```
            return delay(g, x + 1, y, z // 2)
        else:
            return delay(g, x, y + 1, 3 * z + 1)

    return g(0, 0, z)

def go(x: Delayed[T]) -> T:
    raise NotImplementedError("Please implement `go`")

def evaluation(n: int) -> int:
    max_x = 0
    max_y = 0

    for x, y in map(go, (f(i) for i in range(1, n + 1))):
        max_x = max(max_x, x)
        max_y = max(max_y, y)

    return max_x * max_y

if __name__ == "__main__":
    print(f"{evaluation(1)} should produce 0")
    print(f"{evaluation(5)} should produce 10")
    print(f"{evaluation(10)} should produce 78")
```

As a proof of your implementation, please provide what does this call return?

evaluation(100)

Válasz

A helyes válasz:

3225

Magyarázat

The function “delay” wraps the execution of a function in a zero-argument function; this zero argument function is also known as a “thunk.” The function “f” returns the result of “g”, which in turn can either return a thunk or a final value. To fully evaluate this, we just need to keep evaluating the thunks as zero-argument functions until a non-function is returned.

You can reason about this directly from the type annotations: we want a function that takes in a “Delayed[T]” and returns a “T”. A “Delayed[T]” is either a “T”, in which case we simply return it, or a “Callable[[], [Delayed[T]]]”, in which case we can evaluate a zero-argument function and recurse.

3. feladat 0/10 pont

Remove Nones

Below is the code for a function which mutates a given dictionary to remove any keys where the value is None (a common enough thing to do to JSON sometimes).

Since JSON and dictionaries can be nested, we've gone ahead and spruced it up to be able to recursively dive in, and also we wanted to know which keys we removed, which will be populated into the `removed` set.

A "keypath" here is a way of uniquely identifying which part of the nested dictionary was touched, since we could have multiple entries in different sub-dictionaries with the same key. Essentially, it is a tuple of the key at each level to get to the one we actually care about. For instance, if we had this dictionary:

{"foo": {"bar": "baz"}, "qux": "abc"}

the keypath for "bar" (to get to the “baz” value) is ("foo", "bar")

```
def remove_nones(d: dict, keypath: tuple = tuple(), removed: set = set()):
    # Takes a potentially nested dictionary structure and removes
    # all keys where the value is None
    # returns "removed", which lists all the keypaths that were removed

    keys = list(d.keys())
    for k in keys:
        v = d[k]
        if isinstance(v, dict):
            # we need to dive in and remove those too
            remove_nones(v, keypath + (k,), removed)
        elif v is None:
            d.pop(k)
            removed.add(keypath + (k,))
    return removed

d = {1: None, 2: {3: 6, 7: None}, 4: 5}
print(remove_nones(d))
print(d)

d2 = {2: None, 8: {1: 4, 3: None}, 5: 6}
print(remove_nones(d2))
print(d2)
```

However, it's not quite working as intended. For example, if we had this:

```
d = {1: None, 2: {3: 6, 7: None}, 4: 5}
print("removed", remove_nones(d))
print("current dict", d)
```

We'd get as output:

```
removed {(1,), (2, 7)}
current dict {2: {3: 6}, 4: 5}
```

That looks right, but then after that let's try:

```
d2 = {2: None, 8: {1: 4, 3: None}, 5: 6}
print("removed", remove_nones(d))
print("current dict", d)
```

This then prints:

```
removed {(1,), (8, 3), (2,), (2, 7)}
current dict {8: {1: 4}, 5: 6}
```

It says 1 was removed, but it's still there (as it should be). Among the choices here, which line needs to be changed to fix the bug?

Válasz

- [x] Line 1
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Line 6
- [ ] Line 11
- [ ] Line 13

Magyarázat

Some IDEs may point this out naturally, but in Python, default arguments are created once at the time the function is defined. This means that if you use a mutable default, for example an empty list, it creates a single empty list once and continually uses that list whenever no argument is given. This means that any changes to that list from previous calls to the function are there.

While it's possible to make intentional use of this sort of behavior, generally it is to be avoided because it creates easy to miss bugs, which is why many modern IDEs will warn you about something like this, but not every tool will.