TESZTAUTOMATIZÁLÁS

7. forduló

<epam>

A kategória támogatója: EPAM

Ismertető a feladathoz

Felhasznált idő: 00:00/30:00 **Elért pontszám:** 0/65

1. feladat 0/5 pont

A felsoroltak közül mely(ek) valós tesztelési alapelv(ek)?

Válaszok

- [x] Amennyiben a kifejlesztett rendszerek nem használhatóak (vagy nem felelnek meg a felhasználók által támasztott igényeknek, elképzeléseknek), úgy a hibák megtalálása és javítása hasztalan.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] A meg nem talált programhibák esetleges következményei mellett figyelembe kell vennünk azon programhibák következményeit is, melyeket megtaláltunk.
- [x] A meglévő tesztesetek felülvizsgálata, új, eltérő tesztesetek írása szükséges annak érdekében, hogy a rendszer/szoftver különböző részei kerüljenek futtatásra, ezáltal új hibákat találjunk.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] A tesztelésnek, programhibák megtalálásának, nem mindig a programhibák megjavítása a fő célja vagy kívánt eredménye; adatokat gyűjthetünk a működésről, felmérhetjük a rendszer állapotát.

Magyarázat

"Amennyiben a kifejlesztett rendszerek nem használhatóak... ": A hibátlan rendszer téveszméje: valós tesztelési alapelv.

"A meg nem talált programhibák esetleges következményei mellett...": nem valós alapelv.

"A meglévő tesztesetek felülvizsgálata, új, eltérő tesztesetek írása..." : A féregirtó-paradoxon, valós tesztelési alapelv.

"A tesztelésnek, programhibák megtalálásának, nem mindig a programhibák megjavítása a fő célja...": nem valós alapelv.

2. feladat 0/10 pont

Mely válaszok adják meg együtt az összes primitív típust JavaScriptben?

Válaszok

- [ ] boolean, array, string, number
- [x] boolean, number, bigint, string, symbol, null
  Ez a válasz helyes, de nem jelölted meg.
- [ ] object, function, symbol, boolean
- [x] boolean, number, string, undefined
  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

A következő típusok tartoznak a primitív típusokhoz JavaScriptben: null, undefined, boolean, number, string, symbol és bigint.

3. feladat 0/10 pont

Egy alkalmazás készítői szeretnének statisztikát készíteni, hogy a logokban mely területen található a legtöbb probléma. Ehhez szeretnének egy olyan reguláris kifejezést készíteni, ami kigyűjti a logokban található adatokból a fájl nevét, a metódus nevét illetve a sor számát, ahol a hiba található. A példákban ez "at csomag.osztály.metódusnév(fájlnév:sorszám)" formátumban van jelen.

Melyik reguláris kifejezés adja vissza "elfogó csoportokba" (capture group) rendezve a kért adatokat?

Példa log részlet:

```
W/dalvikvm( 1553): threadid=1: uncaught exception
E/( 1553): FATAL EXCEPTION: main
E/( 1553): java.lang.StringIndexOutOfBoundsException
E/( 1553):   at widget.List.makeView(ListView.java:1727)
E/( 1553):   at widget.List.fillDown(ListView.java:652)
E/( 1553):   at widget.List.fillFrom(ListView.java:709)
```

Válasz

- ( ) (\w+)\((\w+\.):(\d+)\)
- (•) (\w+)\(([\w\.]+):([1-9]\d*)\)
  Ez a válasz helyes, de nem jelölted meg.
- ( ) (\w+).(\w+)\.(\w+:\d+)
- ( ) (\w+)\([A-Z][a-z]+[A-Z][a-z]+\.\w+:[1-9]\d+\)

Magyarázat

"(\w+)\((\w+\.):(\d+)\)": nem illeszkedik a fájlnévben a pont utáni részre, nem várja el, hogy a kód sorszámozása 1-ről induljon.

"(\w+)\(([\w\.]+):([1-9]\d+)\)": elvárja, hogy a kód sorszámozása 1-ről induljon, elvárja a fájlnév és sorszám zárójelben létét, fájlnévben lévő pontot, valamint a fájlnév és sorszám közti kettőspont meglétét.

"(\w+).(\w+)\.(\w+:\d+)": nem várja el, hogy a kód sorszámozása 1-ről induljon, nem várja el, hogy a fájlnév és a sorszám zárójelben helyezkedjen el, rosszul definiálja az elfogó csoportokat.

"(\w+)\([A-Z][a-z]+[A-Z][a-z]+\.\w+:[1-9]\d+\)": nem fogadja el a 10 alatti sorszámozású találatokat, amennyiben csak List.java a fájlnevünk kettőnél több tagból áll pl. ListViewSorting, úgy nem lesz rá érvényes a reguláris kifejezés, nem elfogó csoportban adja vissza a fájlnevet és a sorszámot.

4. feladat 0/10 pont

Az alábbiak közül melyik esetben **nem célszerű** automata teszteket írni?

Válasz

- (●) Adatok egyszeri migrálása egy adatbázisból egy másikba
  Ez a válasz helyes, de nem jelölted meg.
- ( ) Egy időpont foglaló alkalmazás fejlesztése
- ( ) Üzleti adatok folyamatos feldolgozását végző rendszer fejlesztése
- ( ) Könyveket árusító webalkalmazás fejlesztése

Magyarázat

Helyes válasz az adatok egyszeri migrálása egy adatbázisból egy másikba, mivel az adott eset csak egyszer kerül végrehajtásra, a teszteket nem használnánk újra és a teszt készítésének költsége nem térülne meg a többi esettel ellentétben.

5. feladat 0/15 pont

A fejlesztők három olyan verziót terveztek egy függvényből, ami három szám közül visszaadja a legkisebbet.

Arra kértek, hogy válaszd ki azt a függvényt amelyiknek a ciklomatikus komplexitása a legalacsonyabb. **Melyiket érdemes választanod?**

A kódok a következőképpen néznek ki:

```
function findMin1(n1, n2, n3) {
        let min;
        if (n1 <= n2 && n1 <= n3){
                min = n1;
        }
        if (n2 <= n1 && n2 <= n3){
                min = n2;
        }
        if (n3 <= n1 && n3 <= n2){
                min = n3;
        }
        return min;
}
```

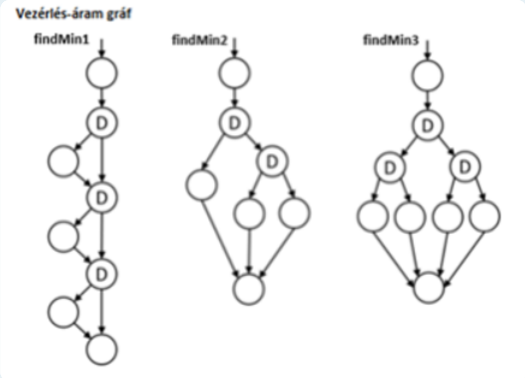
```
function findMin2(n1, n2, n3) {
        let min;
        if (n1 <= n2 && n1 <= n3){
                min = n1;
        } else if (n2 <= n1 && n2 <= n3){
                min = n2;
        } else {
                min = n3;
        }
        return min;
}
```

```
function findMin3(n1, n2, n3) {
        let min;
        if (n1 <= n2) {
                if (n1 <= n3){
                        min = n1;
                } else {
                        min = n3;
                }
        } else {
                if (n2 <= n3){
                        min = n2;
                } else {
                        min = n3;
                }
        }
        return min;
}
```

Válaszok

- [ ] findMin1
- [x] findMin2
  Ez a válasz helyes, de nem jelölted meg.
- [x] findMin3
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Bármelyiket, mert egyforma ciklomatikus komplexitással rendelkeznek

Magyarázat



A ciklomatikus komplexitás definíciója: L – N + 2P, ahol L = az élek/kapcsolatok száma a gráfban, N = a csomópontok száma a gráfban,

P = a nem kapcsolódó részek a gráfban (pl. egy meghívott gráf, vagy szubrutin).

Tehát a findMin1 esetében a ciklomatikus komplexitás: 10-8+2 = 4, findMin2 esetében: 8-7+2=3, findMin3 esetében:11 -9+2 =4

További információ: link.

Kedves Versenyzők!

findMin2 megoldásban helytelenül voltak egyformán kezelve az egyfeltételes és az összetett/többfeltételes "if" utasítások. Aki ezt figyelembe vette, annak a findMin3 adta a legalacsonyabb komplexitást, így mindkét válaszlehetőséget elfogadjuk.

6. feladat 0/15 pont

Az alábbiak közül mely megoldás szimulálja azt, hogy egy 52 lapos francia kártya pakliból (DECK) 4 játékos mindegyikének 5 kártyalapot osztunk?

A francia kártya pakli 4 "színt", valamint színenként 4 figurás lapot és 9 számozott lapot (2-től 10-ig) tartalmaz, így jön ki a 4 * 13 = 52 lap.

Válasz

```
import random
import itertools


COLORS = ["Hearts", "Diamonds", "Clubs", "Spades"]
RANKS = ["Jack", "Queen", "King", "Ace"]
NUMBERS = [str(number) for number in range(2, 11)]




DECK = [f"{item[0]} of {item[1]}" for item in itertools.product(NUMBERS + RANKS, COLORS)]




def deal(count):
    global DECK
    hand = random.sample(DECK, k=count)
    DECK = list(set(DECK) - set(hand))
    return hand


player1_hand = deal(5)
player2_hand = deal(5)
player3_hand = deal(5)
player4_hand = deal(5)
```

Ez a válasz helyes, de nem jelölted meg.

```
import random
import itertools


COLORS = ["Hearts", "Diamonds", "Clubs", "Spades"]
RANKS = ["Jack", "Queen", "King", "Ace"]
NUMBERS = [str(number) for number in range(2, 11)]




DECK = [f"{item[0]} of {item[1]}" for item in itertools.permutations(NUMBERS + RANKS +  COLORS)]




def deal(count):
    global DECK
    hand = random.choices(DECK, k=count)
    DECK = list(set(DECK) - set(hand))
    return hand
```

```
player1_hand = deal(5)
player2_hand = deal(5)
player3_hand = deal(5)
player4_hand = deal(5)
```

```
import random
import itertools


COLORS = ["Hearts", "Diamonds", "Clubs", "Spades"]
RANKS = ["Jack", "Queen", "King", "Ace"]
NUMBERS = [str(number) for number in range(2, 11)]




DECK = [f"{item[0]} of {item[1]}" for item in itertools.product(NUMBERS + RANKS, COLORS)]




def deal(count):
    hand = random.sample(DECK, k=count)
    return hand


player1_hand = deal(5)
player2_hand = deal(5)
player3_hand = deal(5)
player4_hand = deal(5)
```

```
import random
import itertools

COLORS = ["Hearts", "Diamonds", "Clubs", "Spades"]
RANKS = ["Jack", "Queen", "King", "Ace"]
NUMBERS = [str(number) for number in range(2, 11)]


DECK = [f"{item[0]} of {item[1]}" for item in itertools.product(NUMBERS + RANKS, COLORS)]


def deal(count):
    global DECK
    hand = random.choices(DECK, k=count)
    DECK = DECK - hand
    return hand

player1_hand = deal(5)
player2_hand = deal(5)
player3_hand = deal(5)
player4_hand = deal(5)
```

Magyarázat

A pakli előállításához az itertools modul product függvénye a megfelelő, mivel a "színek" és a számok + figurák listáját kell összekombinálnunk, a permutations függvény nem a kívánt eredményt adná.

Egy listából ismétlődés nélkül a random modul sample függvényével tudunk véletlenszerűen elemeket kiválasztani, a choices esetében előfordulhat ismétlődés.

A sample azonban önmagában nem veszi ki a kiválasztott elemeket az eredeti listából, így nekünk kell gondoskodnunk azok kivételéről a pakliból. Listából listát egyszerűen, egy kivonás operátorral nem lehet eltávolítani, erre számos megoldás van, egy viszonylag egyszerű a halmazzá konvertálás után történő kivonás.

Összességében így az a megoldás jó, amiben itertools.producttal állítjuk elő a paklit, random.sample-lel választjuk ki a kártyákat, és halmaz konverzióval vesszük ki ezeket a pakliból.