TESZTAUTOMATIZÁLÁS

6. forduló

<epam>

A kategória támogatója: EPAM

Ismertető a feladathoz

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

1. feladat 0/5 pont

Melyik URI query stringnek helytelen a formátuma?

Válasz

- ```
  http://www.weboldal.hu/oldal.html?parameter=parameter1
  ```
- ```
  http://www.weboldal.hu/oldal.html?parameter=parameter1?parameter2=parameter2
  ```

  Ez a válasz helyes, de nem jelölted meg.
- ```
  http://www.weboldal.hu/oldal.html?parameter=parameter1&parameter2=parameter2
  ```
- ```
  http://www.weboldal.hu/oldal.html?parameter=parameter+string+value
  ```

Magyarázat

?parameter=parameter1 - Megfelelő formátumú query string, helyesen "?"-lel kezdődik és egy paramétert tartalmaz.

?parameter=parameter1?parameter2=parameter2 - A helyes válasz ez, mivel két "?"-et tartalmaz. A második paraméter előtt "&" jelet kell használni.

?parameter=parameter1&parameter2=parameter2 - Megfelelő formátumú query string, "?"-lel kezdődik és két paramétert tartalmaz "&" jellel elválasztva.

?parameter=parameter+string+value - Megfelelő formátumú query string, "?"-lel kezdődik és egy paramétert tartalmaz.

2. feladat 0/10 pont

Az alábbi funkciók közül melyeket támogatja a TestNG keretrendszer és a JUnit keretrendszer is?

Válaszok

- [ ] Dependency test
- [x] Exception test
  Ez a válasz helyes, de nem jelölted meg.
- [x] Suite test
  Ez a válasz helyes, de nem jelölted meg.
- [x] Parameterized test
  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

Csak a TestNG támogatja a dependency testet. Ezzel a funkcióval megadható, hogy egy teszt metódus csak akkor fusson le, ha a beállított teszt metódus már lefutott sikeresen.

3. feladat 0/10 pont

Tegyük fel, olyan cégnél dolgozol, amely szoftvermegoldásokat kínál orvosi rendszerekhez. Jelenleg te azt a szoftverkomponenst teszteled, amely a lélegeztetőgépeket működteti és szabályozza a tüdőbe juttatott levegő oxigén koncentrációját. A vizsgált modul 40 független atomi feltételből áll.

Fehérdobozos (white-box) tesztelést kell végezned ehhez a modulhoz, és várhatóan két hónapon belül be kell fejezned a tesztelést.

Melyik fehérdobozos vizsgálati technikát választod ehhez a forgatókönyvhöz?

Válasz

- (•) MC/DC (modified condition/decision coverage) tesztelés
  Ez a válasz helyes, de nem jelölted meg.
- ( ) MC (multiple condition) tesztelés
- ( ) Döntési tesztelés
- ( ) UI tesztelés (user interface)

Magyarázat

Biztonság kritikus rendszerről van szó. A döntési tesztelés nem alkalmas biztonság kritikus rendszerek tesztelésére.

Teljes MC (multiple condition) tesztelés nem javasolt és nehezen kivitelezhető (2 hónapon belül) tekintve hogy 2^40 darab tesztesetre lenne szükség így az MC/DC tesztelés a megfelelő választás mivel átfogóbb mint a döntési tesztelés viszont csak 41 darab tesztesetre van hozzá szükség szemben a 2^40 darab tesztesettel. A UI tesztelés nem releváns ebben az esetében ugyanis a UI tesztelésnek nem célja az atomi feltételek kiértékelése.

4. feladat 0/10 pont

A következő állítások közül melyik **nem igaz** az automata tesztekre?

Válasz

- ○ Hatékonyan növeli a teszt lefedettséget
- ○ Gyorsabb visszajelzést ad az alkalmazás minőségéről, mint a kézi tesztelés
- ● Időt spórolunk meg vele rövidtávon a kézi teszteléshez képest
  Ez a válasz helyes, de nem jelölted meg.
- ○ Megbízhatóbb, a kézi teszteléshez viszonyítva

Magyarázat

Helyes válasz: időt sprólunk meg vele rövidtávon a kézi teszteléshez képest. Valóban időt spórolunk meg az automata tesztekkel, de csak hosszútávon, mivel az automata teszteknek kezdetben magas előkészítési költsége van.

5. feladat 0/15 pont

Adott a következő DOM struktúra. Mely selectorokkal tudjuk kiválasztani azt a model tag-et, aminek a szövege Prius és a vele egy szinten lévő year tag szövege 2000?

```
<vehicle>
        <cars>
                <car>
                        <brand id="1">Toyota</brand>
                        <model>Prius</model>
                        <year>2000</year>
                </car>
                <car>
                        <brand id="1">Toyota</brand>
                        <model>Prius</model>
                        <year>2015</year>
                </car>
        </cars>
</vehicle>
```

Válaszok

- ☑ //*[year='2000']/ancestor-or-self::car/model
  Ez a válasz helyes, de nem jelölted meg.
- ☑ //*[year='2000']/model
  Ez a válasz helyes, de nem jelölted meg.
- ☐ //model[text()='Prius']/following-sibling::year[text()='2000']
- ☑ //brand[contains(text(),'Toyota')]/following-sibling::model[text()='Prius']/following-sibling::year[text()='2000']/preceding-sibling::model
  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

A //*[year='2000']/ancestor-or-self::car/model selector helyes, először megkeresi azt a csomópontot, amihez tartozik year 2000 érték, majd keresi azon belül (vagy ősében) a car-hoz tartozó modelt.

A //*[year='2000']/model selector helyes, először megkeresi azt a csomópontot, amihez tartozik year 2000 érték, majd keresi az azon belüli modelt.

A //model[text()='Prius']/following-sibling::year[text()='2000'] selector helytelen válasz, mert a 2000 értékkel rendelkező yeart fogja elkapni, nem pedig a modelt.

A //brand[contains(text(),'Toyota')]/following-sibling::model[text()='Prius']/following-sibling::year[text()='2000']/preceding-sibling::model selectorban sok a szükségtelen szűrés, de helyes, először megkeresi azt a brand-et aminek a szövege tartalmazza a Toyota-t, majd ennek az utána következő testvér elemét, ami Prius szövegű model, majd az utána következő testvérelemét, ami 2000 értékű year és végül a year-t megelőző model elemre lép vissza.

6. feladat 0/15 pont

Az alábbi kódrészletek közül melyekkel teszteljük, hogy egy exception helyesen dobódik-e?

Válasz

```
@Test
        public void testMethod() {
                try {
                        new MyClass().method1();
                } catch(Exception e) {
                        assertTrue(true);
                } finally {
                        fail();
                }
        }
```

```
@Test(expected=NullPointerException.class)
        public void testMethod() {
                new MyClass().method1();
        }
```

Ez a válasz helyes, de nem jelölted meg.

```
@Test(expectedException=NullPointerException.class)
        public void testMethod() {
                new MyClass().method1();
        }
```

```
@Test
        public void testMethod() {
                try {
                        new MyClass().method1();
                } catch(Exception e) {
                        fail();
                }
        }
```

Magyarázat

Mind a TestNG, mind a JUnit támogatja standard annotation használatát az elvárt exception tesztelésre.

- TestNG: @Test(expectedException=NullPointerException.class)

- JUnit: @Test(expected=NullPointerException.class)

A 'fail' metódus hívással tudjuk kikényszeríteni egy teszt bukását, így annak catch ágban való hívása pont az ellenkező eredményt hozza.

A finally ág mindig lefut, így az a teszteset folyamatosan bukik.