به نام خدا

تمرین چهارم درس آزمون نرم افزار

آریا وارسته نژاد

9531345

آریا خلیق

9524014

حسین محمدی

9533081

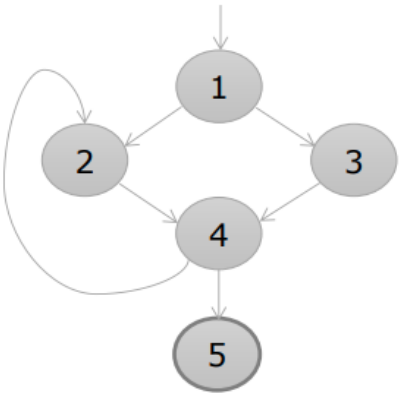
1400

```
def is_extendable(path, graph):
    can_reach_end = True
    later_nodes = graph['edges'][path[-1]]
    for n in later_nodes:
        if n not in path or n == path[0]:
            can_reach_end = False
            break

    if is_prime(path, graph) or can_reach_end:
        return False
    return True


def find_simple_paths(graph, ex_paths, paths):
    for p in ex_paths:
        if is_prime(p, graph):
            paths.extend([p])

    new_ex_paths = []
    for p  in ex_paths:
        if is_extendable(p , graph):
            new_ex_paths += [p]
    ex_paths = new_ex_paths

    new_ex_paths = []
    for p in ex_paths:
        for nx in graph['edges'][p[-1]]:
            if nx not in p or nx == p[0]:
                new_ex_paths.append(p + (nx,))

    if len(new_ex_paths) > 0:
        find_simple_paths(graph, new_ex_paths, paths)


def is_prime(path, graph):
```

```
    can_reach_end = True
    later_nodes = graph['edges'][path[-1]]
    for n in later_nodes:
        if n not in path or n == path[0]:
            can_reach_end = False
            break

    can_reach_head = True
    former_nodes = []

    for n in graph['nodes']:
        if path[0] in graph['edges'][n]:
            former_nodes += [n]

    for n in former_nodes:
        if n not in path or n == path[-1]:
            can_reach_head = False

    if len(path) >= 2 and path[0] == path[-1] or
can_reach_head and can_reach_end:
        return True
    return False


def find_prime_paths(graph):
    ex_paths = [(n,) for n in graph['nodes']]
    simple_paths = []

    find_simple_paths(graph, ex_paths, simple_paths)
    prime_paths = sorted(simple_paths, key=lambda a:
(len(a), a))
    for p in prime_paths:
        print(list(p))


graph = {
```

```
    'nodes': [1, 2, 3, 4, 5],
    'edges': {
        1: [2, 3],
        2: [4],
        3: [4],
        4: [2, 5],
        5: []
    }
}
find_prime_paths(graph)
```

ورودی گراف این برنامه طبق مثال اسلایدها تنظیم شده است:



Simple Paths: [1,2,4,5], [1,3,4,2], [1,3,4,5], [1,2,4], [1,3,4], [2,4,2], [2,4,5], [3,4,2], [3,4,5], [4,2,4], [1,2], [1,3], [2,4], [3,4], [4,2], [4,5], [1], [2], [3], [4], [5]

Prime Paths: [1,2,4,5], [1,3,4,2], [1,3,4,5], [2,4,2], [4,2,4]

(2

برنامه‌ای که انتخاب کرده‌ایم کدهای بکاند یک وب‌سایت است که به زبان پایتون و با فریمورک جنگو توسعه داده شده و دارای یونیت‌تست برای بخش‌های مختلف نرم‌افزار است. از ابزار Caverage برای مشخص کردن میزان کاورج تست‌ها استفاده می‌کنیم که مناسب ارزیابی برنامه‌های پایتونی است. همچنین در CI و در Gitlab این خروجی کاورج به صورت badge در ریپازیتوری آپدیت می‌شود. تکه کدهای اجرای کاورج و گرفتن خروجی آن به صورت زیر است.

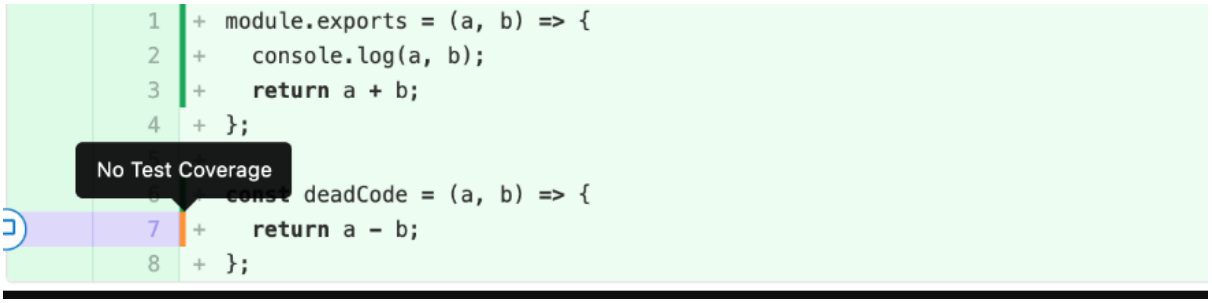
```
- coverage run --source='api' manage.py test  -v 2 --pattern *test*.py
- coverage report
- coverage xml
```

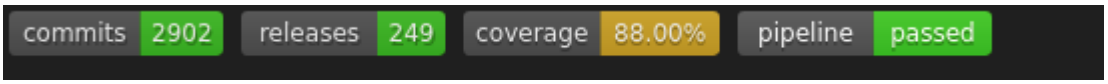
خود برنامه coverage دستور manage.py test *test*.py را اجرا می‌کند و به این روش تعداد خط‌های اجرا شده در هنگام اجرا شدن تست‌ها را مشخص می‌کند.

خروجی coverage report به صورت زیر است:

```
arya@arya:~/projects/axya_api$ coverage report
Name                                                                                                 Stmts   Miss  Cover
------------------------------------------------------------------------------------------------------------------
api/__init__.py                                                                                          9      0   100%
api/admin.py                                                                                           281     53    81%
api/apps.py                                                                                              8      0   100%
api/cad/__init__.py                                                                                      0      0   100%
api/cad/cad_files_serializer.py                                                                         12     12     0%
api/cad/cad_service.py                                                                                  54     17    69%
api/cad/test_cad_service.py                                                                             28      0   100%
api/celery.py                                                                                           20      0   100%
api/company/__init__.py                                                                                  0      0   100%
api/company/acl_company_service_test.py                                                                 43      0   100%
api/company/capabilities/__init__.py                                                                     0      0   100%
api/company/capabilities/capability_serializer.py                                                       50      4    92%
api/company/capabilities/capability_service.py                                                          50      9    82%
api/company/capabilities/test_capability_service.py                                                     58      0   100%
api/company/company_decorators.py                                                                       29      0   100%
api/company/company_file_references/__init__.py                                                          0      0   100%
api/company/company_file_references/company_file_references_serializer.py                                6      0   100%
api/company/company_file_references/company_file_references_service.py                                  51      7    86%
api/company/company_file_references/test_company_file_references_service.py                             25      0   100%
```

از دستور coverage xml می‌توان برای ایجاد خروجی xml مطابق با استاندارد covertura استفاده کرد که در Merge Request داخل Gitlab به شکل زیر نشان داده می‌شود:





خط‌های سبز نشان‌دهنده اجرا شدن حداقل یکباری آن‌ها است. توجه شود که برای ایجاد این باید خروجی xml را به صورت report در آریتیفکت این‌گونه تعریف کرد که کیت‌لب بتواند آن را شناسایی کند:

```
artifacts:
  reports:
    cobertura: coverage.xml
```

ابزارهای Test Coverage میزان خط‌هایی که در اجرای تست‌ها اجرا می‌شوند را تقسیم بر تعداد خطوط کل برنامه می‌کنند و به این صورت درصد تست کاورج به دست می‌آید.

از مهم‌ترین مشکلات این ابزارها این است که می‌توان عملا تست‌های قوی‌ای نداشت اما به تست کاورج ۱۰۰ درصد رسید. چیزی که باید در این موارد حواسمان باشد این است که باید تست‌ها قوی و حالت‌های مختلف منطق بیزنسی، تکه‌های کد و ... را پشتیبانی کنند.

3-الف)

این مدل چهار حالته میباشد

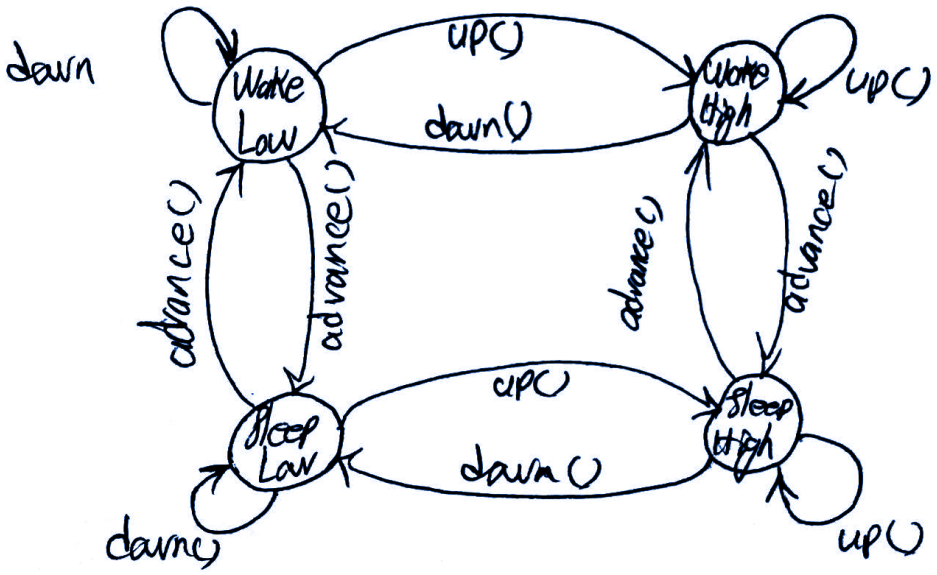
Wake, Low

Wake, High

Sleep, Low

Sleep, High

3-ب)



3-ج)

حالت Wake, Low حالت آغازین ما است.

down(), up(), up(), down(), advance(), down(), up(), up(), advance(), advance(), down(), advance().

4)

مورد اول

کارت به کارت

برای انجام این عمل مشترک، پس از وارد کردن کارت بانکی خود به خودپرداز و وارد کردن رمز عبورش میتواند مبلغی زیر سقف مجاز برای کارت به کارت انتخاب کرده و برای کارت مقصد مورد نظر خود واریز کند.

تست کیس ها

- کارت را به صورت درست وارد کند. هم جهت ها و هم برعکس وارد نکند.
- صحت رمز عبور وارد شده از سوی کاربر.
- مجاز بودن اطلاعات کارت مثل تاریخ انقضاء و ....
- اگر کاربر بیش از تعداد دفعات مجاز خطا در وارد کردن رمز عبور داشت، خوردن کارت

:]

- انتخاب کردن کارت به کارت از منو مربوطه.
- ثبت مبلغ مورد نظر.
- زیر سقف مجاز بودن مبلغ وارد شده.
- در صورت ورود مبلغ بالا نمایش خطای مناسب.
- بررسی این که آیا این میزان پول در حساب مشتری وجود دارد.
- در صورت موجود نبودن آن مقدار پول نمایش خطای مناسب.
- برسی اعتبار سریال 16 رقمی مربوط به کارت مقصد.
- استعلام این که کارت معبتر بوده (تاریخ انقضاء کارت مقصد فرا نرسیده باشد).
- نمایش تائید اطلاعات.
- نمایش صحیح کد رهگیری پرداخت.
- چاپ رسید مناسب.
- پرسش برای این که آیا کار دیگری مشتری با خودپرداز دارد.
- پس دادن کارت.
- رفتن به حالت انتظار برای ورود کارت بعدی.

مورد دوم

استفاده از ATM جهت پرداخت کردن قبوض

برای انجام این عمل مشترک، پس از وارد کردن کارت بانکی خود به خودپرداز و وارد کردن رمز عبورش میتواند از منو موجود در خودپرداز پرداخت قبوض را انتخاب نماید. پس از این دو شیوه برای او قابل انتخاب است.

روش اول: استفاده از بارکدخوان

روش دوم: استفاده از شناسه قبض و شناسه پرداخت

در روش اول پس از قرار دادن قسمت بارکد قبض در بخش مخصوص بارکدخوان، به صورت خودکار بارکد اسکن شده و اطلاعات قبض شامل نام مشترک و مبلغ و... به نمایش در آمده و در صورت صحت اطلاعات او میتواند ادامه داده و پرداخت خود را کامل کند. در روش دوم هم به جای بارکد خوان، مشتری باید شناسه قبض و شناسه پرداخت را وارد کرده و مانند حالت قبلی اطلاعات را مشاهده و تائید کند و پرداخت را تکمیل نماید.

تست کیس ها:

- درست وارد کردن کارت.
- صحت رمز عبور وارد شده از سوی کاربر.
- مجاز بودن اطلاعات کارت مثل تاریخ انقضاء و ....
- اگر کاربر بیش از تعداد دفعات مجاز خطا در وارد کردن رمز عبور داشت، خوردن کارت :]
- صحت عملکرد صفحه نمایش.
- منطبق بودن عملکرد دکمه ها با آنچه در نمایشگر به مشتری نمایش داده می شود.
- نمایش خطای مناسب در صورت وارد کردن شناسه قبض و شناسه پرداخت نادرست.
- بررسی این که آیا این میزان پول در حساب مشتری وجود دارد.
- نمایش دادن شماره پیگیری.
- تست کردن فیش خروجی.
- تست درست پس دادن کارت بانکی.
- رفتن به حالت انتظار برای ورود کارت بعدی.