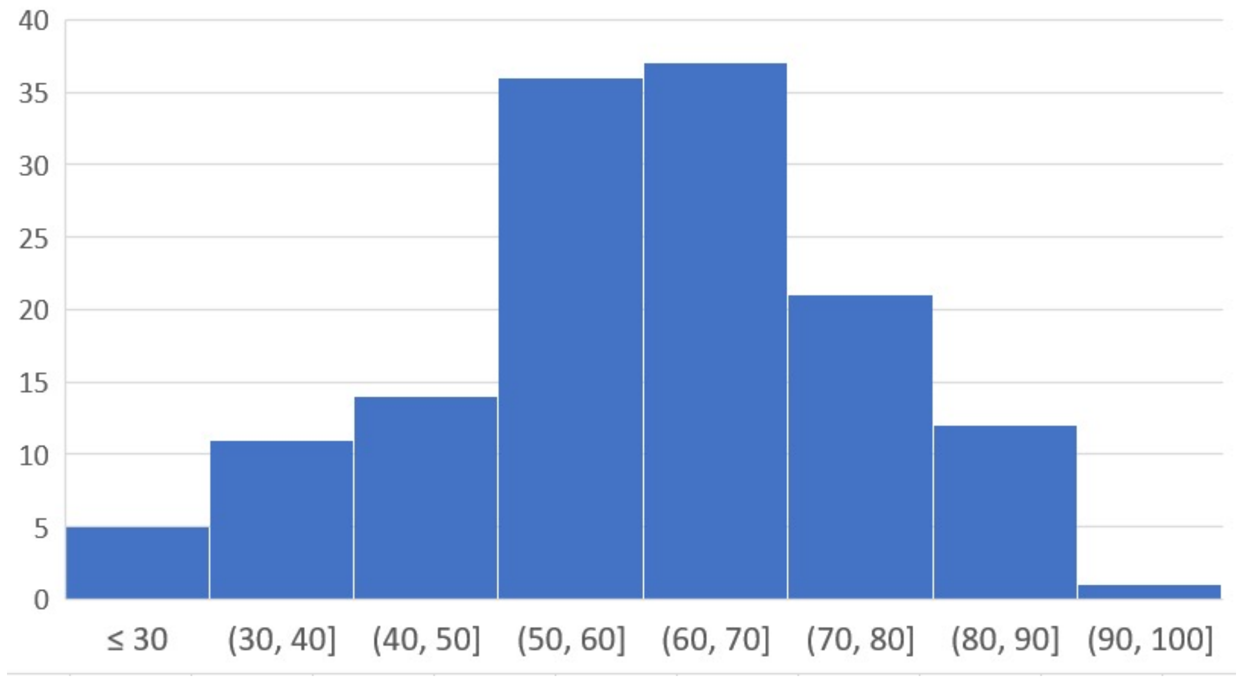
Review of Midterm Exam

考试情况

- 90分以上：1人
- 80分以上：13人
- 60分以上：71人

- 平均分：60



题目	1	2	3	4	5	6
满分	11	26	15	10	16	22
平均	7.14	17.03	13.01	6.77	5.44	9.84
考生认为的难度	😓	😓	🤧	🤧	🥺	🥺
助教认为的难度	😁	😁	😁	😁	🤧	😓

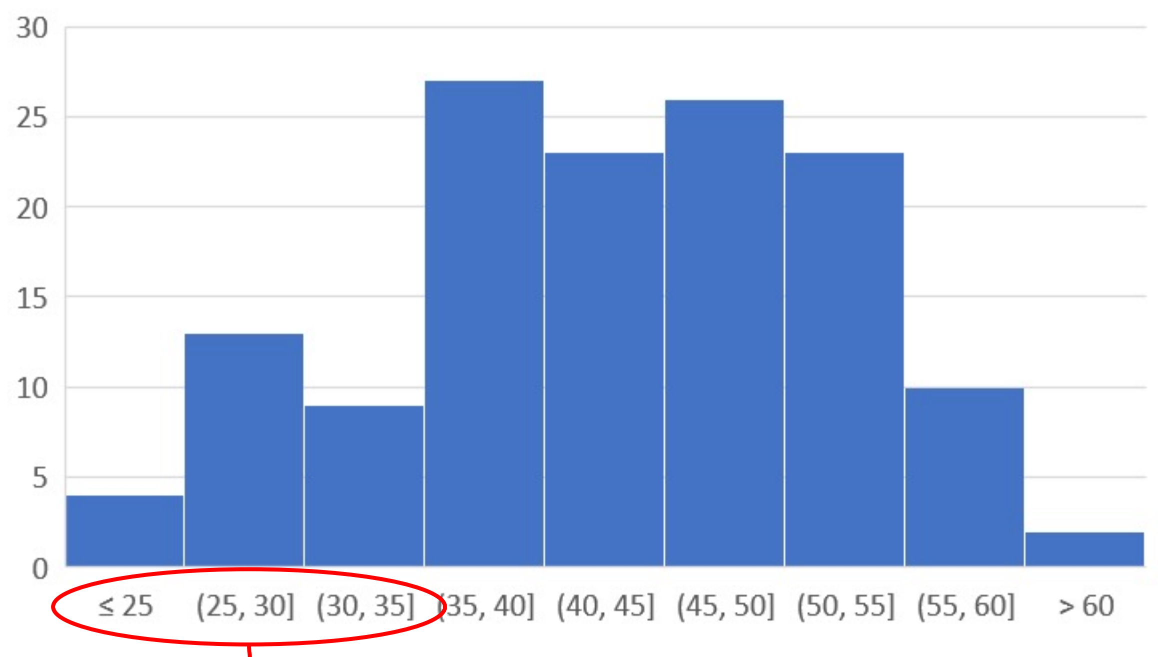
最后两题太难了，
我们来看看去掉这
两题之后的情况

考试情况

- 只看前四题（总分62分）

- 60分以上：2人
- 37分以上：102人

- 平均分：43（大约为70%）

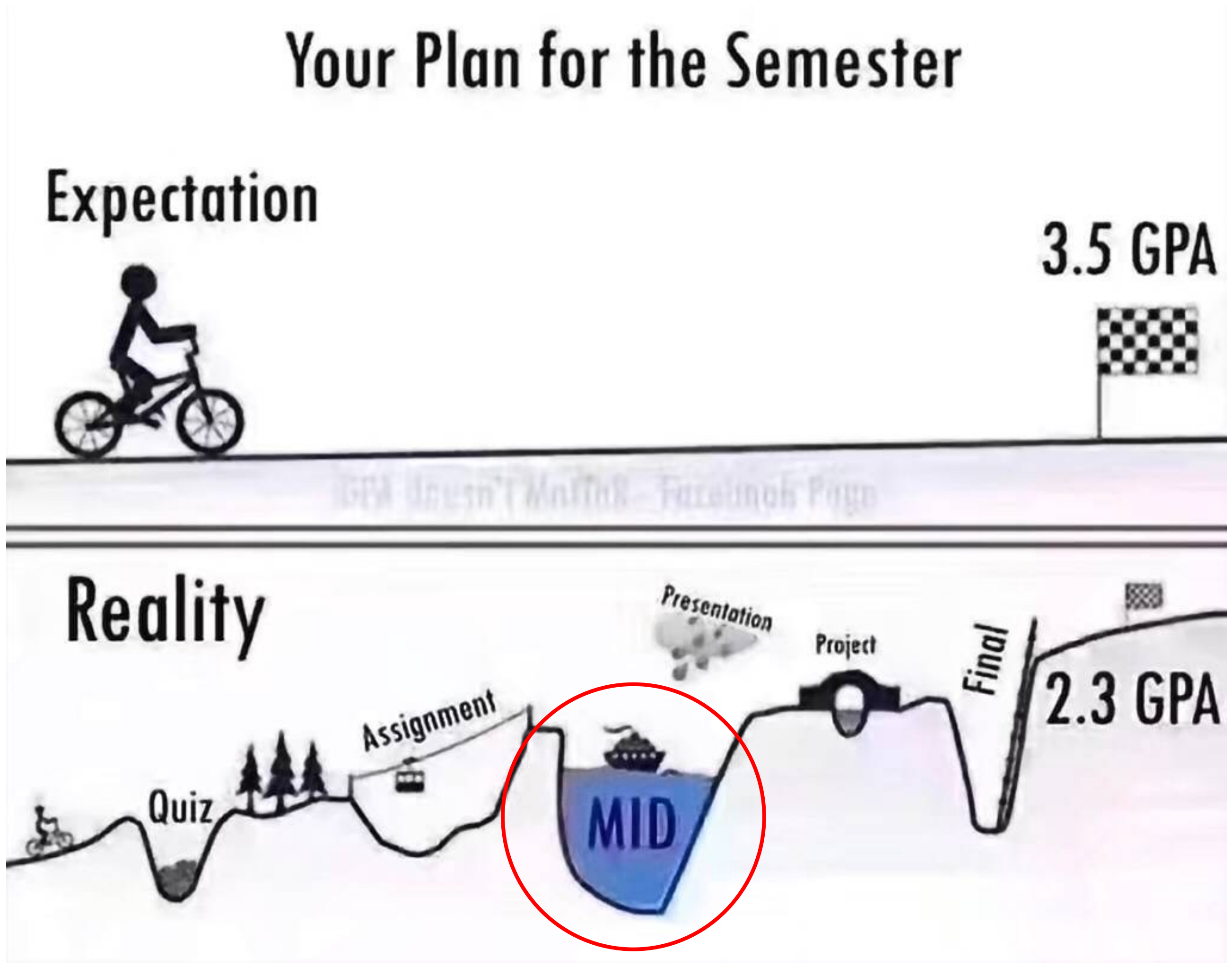


题目	1	2	3	4
满分	11	26	15	10
平均	7.14	17.03	13.01	6.77
考生认为的难度	😓	😓	🤮	🤮
助教认为的难度	😁	😁	😁	😁

这些同学要加把劲了，
不然期末考试送分都送不到！

考试感想

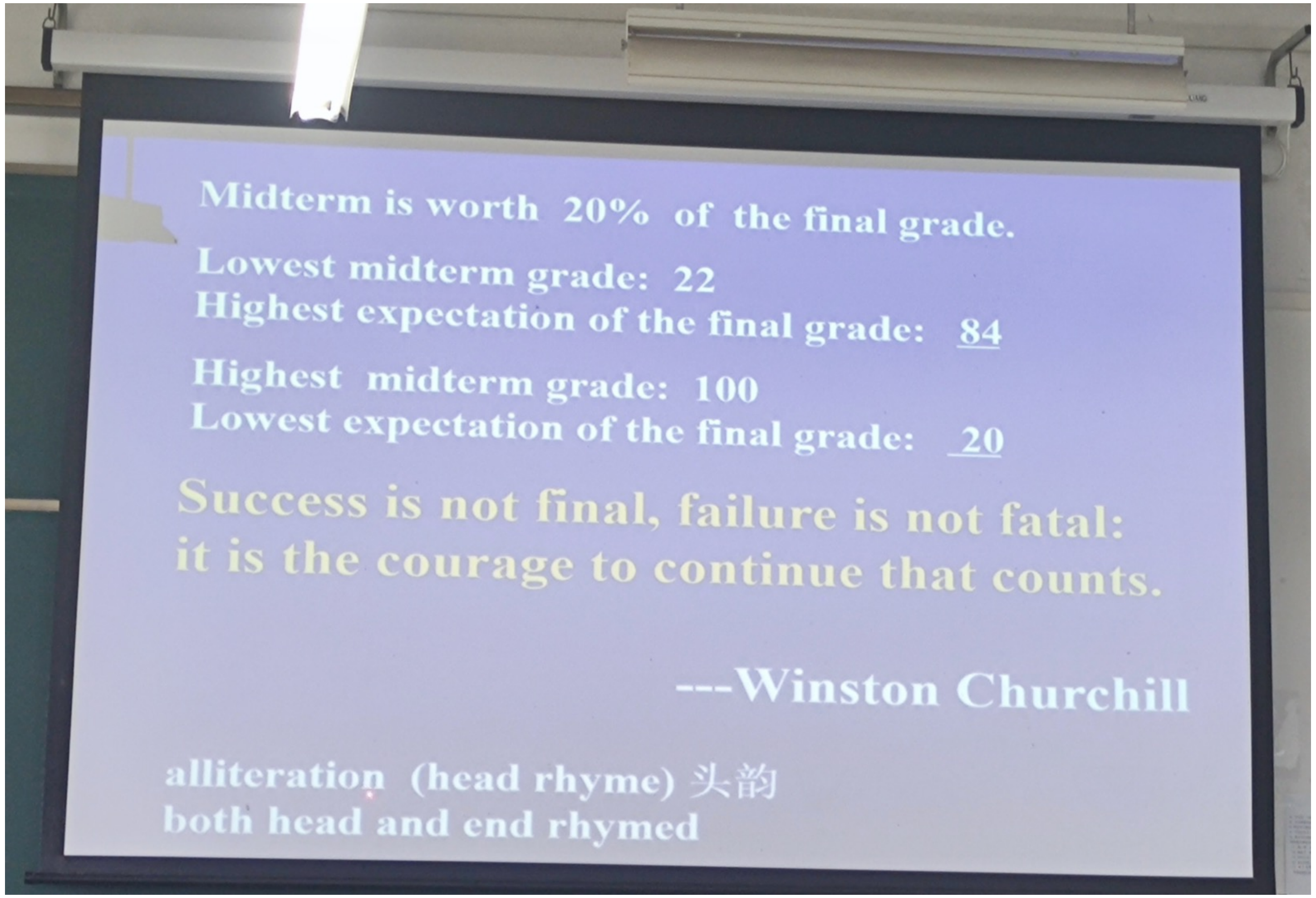
- 对同学们而言：
  - 大学没那么简单



大部分同学现在的处境

考试感想

- 对同学们而言：
  - 大学没那么简单
  - 但是也没有那么恐怖



Midterm is worth 20% of the final grade.

Lowest midterm grade: 22
Highest expectation of the final grade: 84

Highest midterm grade: 100
Lowest expectation of the final grade: 20

Success is not final, failure is not fatal:
it is the courage to continue that counts.

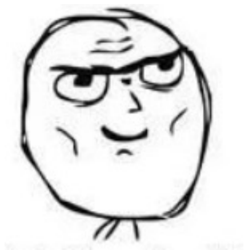
---Winston Churchill

alliteration (head rhyme) 头韵
both head and end rhymed

——《大学物理》

考试感想

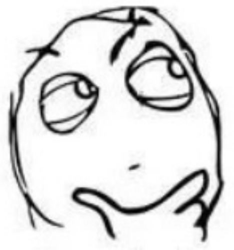
- 对同学们而言：
  - 大学没那么简单
  - 但是也没有那么恐怖
- 对于助教而言：
  - 送分没那么简单



助教

```
if some_condition:
    return subseq(______, ______)
return subseq
```

都这么提示了，同学们肯定能想到
下面一空也是递归吧



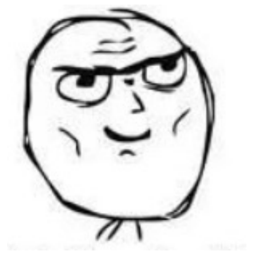
同学们

```
if some_condition:
    return subseq(______, ______)
return False
```

都这么提示了，
下面一空肯定不可能是递归

考试感想

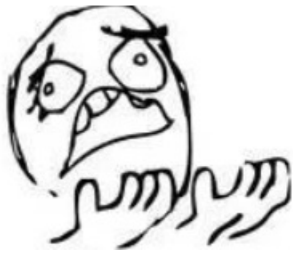
- 对同学们而言：
  - 大学没那么简单
  - 但是也没有那么恐怖
- 对于助教而言：
  - 送分没那么简单
  - 送分真没那么简单

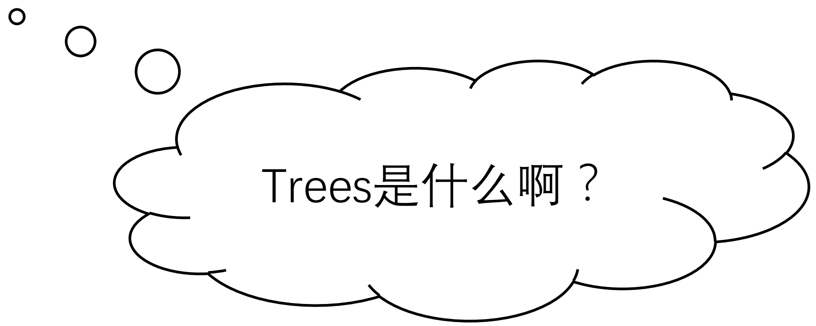

助教

Tree
(单数)

Trees
(复数)

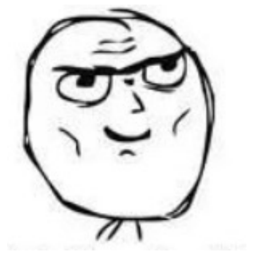
Tree


同学们



考试感想

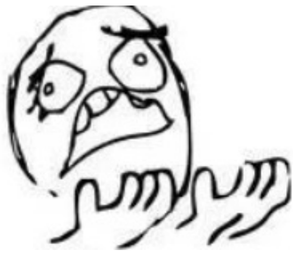
- 对同学们而言：
  - 大学没那么简单
  - 但是也没有那么恐怖
- 对于助教而言：
  - 送分没那么简单
  - 送分真没那么简单
  - 有点 “高估” 了同学们的能力



助教

Tree
（单数）

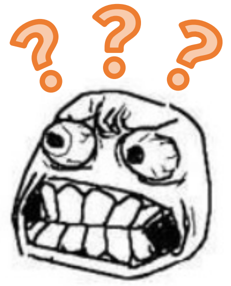
Trees
（复数）



同学们

Tree

Trees
(tree of tree)

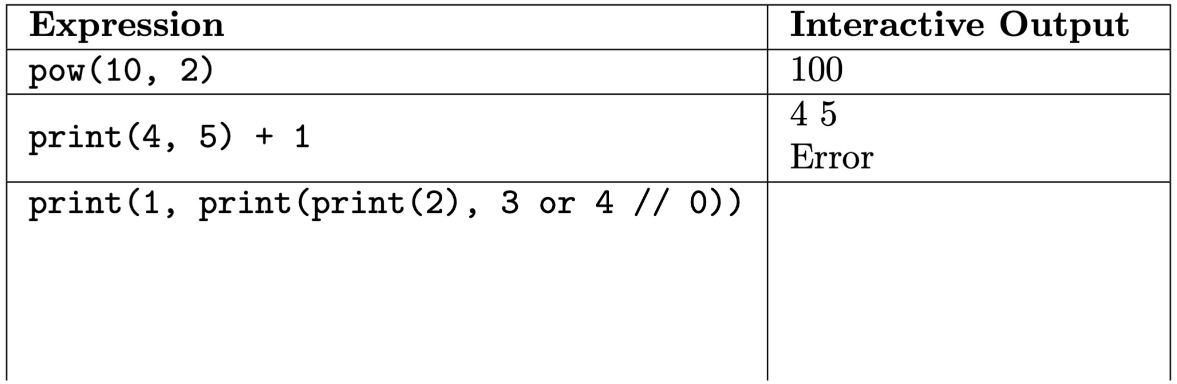


助教
（批卷子时）

Problem 1: Book Me on Weekdays

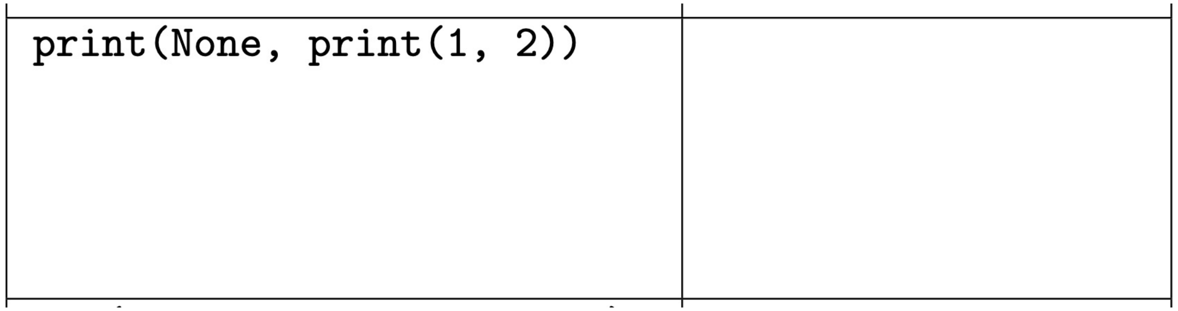
```
>>> print(print(2), 4)
???


>>> len([1, 2, 3, [4, 5]])
???

>>> range(10)[2]
???
```

Problem 1: Book Me on Weekdays



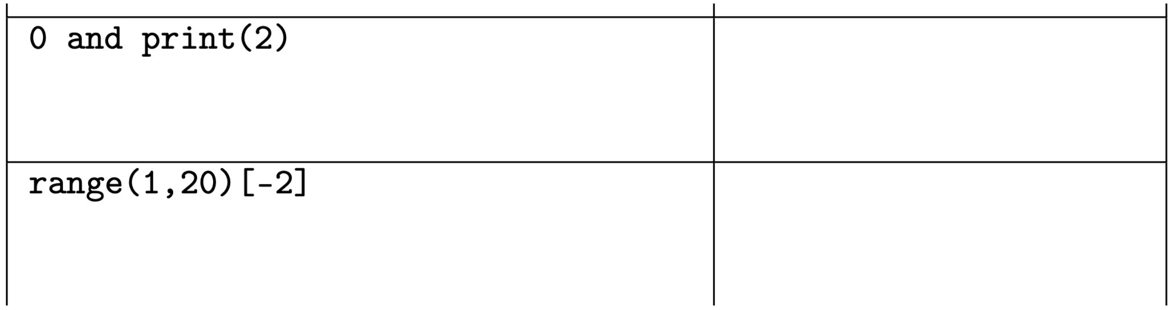
Expression	Interactive Output
pow(10, 2)	100
print(4, 5) + 1	4 5 Error
print(1, print(print(2), 3 or 4 // 0))	

Spring 2019



print(None, print(1, 2))	

Fall 2018

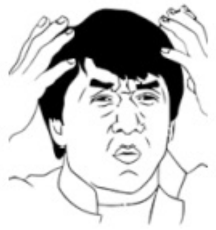


0 and print(2)	
range(1,20)[-2]	

Spring 2017

Problem 1: Book Me on Weekdays

```
>>> print(print(2), 4)
2
None 4
```

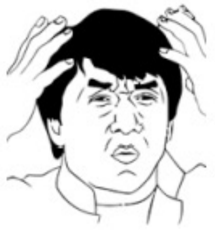


错误答案：

```
Error

2
Error

2
Function 4
```

Problem 1: Book Me on Weekdays

```
>>> len([1, 2, 3, [4, 5]])
4
```



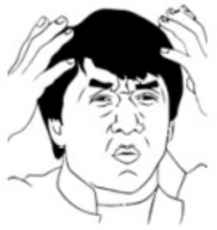
错误答案：

1

5

Problem 1: Book Me on Weekdays

```
>>> range(10)[2]
2
```



错误答案：

```
Error

1

3

[2, 2, 2, 2, ...]
```

Problem 1: Book Me on Weekdays

```
weekday = lambda d: not weekend(d)
weekend = lambda d: \
            d % 7 == 6 or d % 7 == 0
luckday = lambda d, l: d % l == 0

today = 20211110

def future(today):
    if weekend(today):
        print('weekend')
    else:
        yield today
    yield from future(today + 1)

def appoint(dates, cond, act):
    for date in dates:
        if cond(date):
            act = act(date)

def booking(n=0):
    def booked(d):
        print(n + 1, d)
        return booking(n + 1)
    return ok if n >= 3 else booked

ok = lambda ok: booking(ok + 1)
```

```
>>> today is weekday or weekend

>>> False or weekend

>>> weekend
Function
```

Problem 1: Book Me on Weekdays

```
weekday = lambda d: not weekend(d)
weekend = lambda d: \
            d % 7 == 6 or d % 7 == 0
luckday = lambda d, l: d % l == 0

today = 20211110

def future(today):
    if weekend(today):
        print('weekend')
    else:
        yield today
    yield from future(today + 1)

def appoint(dates, cond, act):
    for date in dates:
        if cond(date):
            act = act(date)

def booking(n=0):
    def booked(d):
        print(n + 1, d)
        return booking(n + 1)
    return ok if n >= 3 else booked

ok = lambda ok: booking(ok + 1)
```

```
>>> luckday(today)(2)
Error
```

Problem 1: Book Me on Weekdays

```
weekday = lambda d: not weekend(d)
weekend = lambda d: \
            d % 7 == 6 or d % 7 == 0
luckday = lambda d, l: d % l == 0

today = 20211110

def future(today):
    if weekend(today):
        print(‘weekend’)
    else:
        yield today
    yield from future(today + 1)

def appoint(dates, cond, act):
    for date in dates:
        if cond(date):
            act = act(date)

def booking(n=0):
    def booked(d):
        print(n + 1, d)
        return booking(n + 1)
    return ok if n >= 3 else booked

ok = lambda ok: booking(ok + 1)
```

```
>>> a = [0, 1]
>>> b = iter(a)
>>> for x in a:
...     for y in b:
...         print(x, y)
0 0
0 1
```

Problem 1: Book Me on Weekdays

```
weekday = lambda d: not weekend(d)
weekend = lambda d: \
            d % 7 == 6 or d % 7 == 0
luckday = lambda d, l: d % l == 0

today = 20211110

def future(today):
    if weekend(today):
        print('weekend')
    else:
        yield today
    yield from future(today + 1)

def appoint(dates, cond, act):
    for date in dates:
        if cond(date):
            act = act(date)

def booking(n=0):
    def booked(d):
        print(n + 1, d)
        return booking(n + 1)
    return ok if n >= 3 else booked

ok = lambda ok: booking(ok + 1)
```

print不是yield

```
>>> f = future(4)
>>> for i in range(3):
...     print(next(f))
4
5
weekend
weekend
8
```

Problem 1: Book Me on Weekdays

```
weekday = lambda d: not weekend(d)
weekend = lambda d: \
            d % 7 == 6 or d % 7 == 0
luckday = lambda d, l: d % l == 0

today = 20211110

def future(today):
    if weekend(today):
        print('weekend')
    else:
        yield today
    yield from future(today + 1)

def appoint(dates, cond, act):
    for date in dates:
        if cond(date):
            act = act(date)

def booking(n=0):
    def booked(d):
        print(n + 1, d)
        return booking(n + 1)
    return ok if n >= 3 else booked

ok = lambda ok: booking(ok + 1)
```

```
booking(1)
booked(2)
booked(3)
ok(4)
ok(5)
...
```

```
>>> x = [1, 2, 3, 4, 5, 6, 7]
>>> y = weekday
>>> z = booking
>>> appoint(x, y, z)
2 2
3 3
Function
```

Problem 2: A Tale of Two Diagrams

送分题，不讲了

容易扣分的点（nonlocal）：
- f1里面的f应该指向f3的λ
- global里的st应该指向f5的λ

整整26分应该都是白给的

Problem 3: Subsequence

送！分！题！

```
def subseq(l, s):
    if s == []:
        return _True  (2分)________________________________________________
    elif l == []:
        return _False  (2分)_______________________________________________
    elif _l[0] == s[0]  (2分)______________________________________________:
        return subseq(_l[1:]  (1分)_________________, s[1:]  (1分)________________)
    return subseq(l[1:], s)  (2分)__________________________________________
```

Problem 3: Subsequence

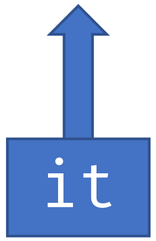
送！分！题！

```
def subseq(l, s):
    def helper(it, target):
        for value in it:
            if __value == target  (1分)__________________________________________:
                return True
        return False
    it = iter(__l  (1分)______)
    return all([_helper(it, v)  (2分)_______________________ for v in __s  (1分)____])
```

Problem 3: Subsequence

How it works?

```
l = [1, 2, 3, 4, 5, 6]
```



```
s = [2, 4, 6, 3]

all([ True True True False])
```

```
for value in it:
    if __value == target  (1分)__
        return True
return False
```

Problem 4: Announce Highest

送！分！题！

```
def announce(n):
    return _detect(0)    (2分)_____________________________________________(n)
def detect(max_value):
    def helper(cur_value):
        if cur_value > max_value    (1分)__________________________________________:
            print(cur_value)    (1分)__________________________________________
        return detect(max(cur_value, max_value))    (2分, detect 1分, max 1分)__
    return helper
```

Problem 4: Announce Highest

送！分！题！

```
def increasing_subsequence(l):
    """
    >>> increasing_subsequence([1, 2, 2, 4, 3, 5])
    1
    2
    4
    5
    """

    f = _announce    (1分)____________________________________________________

    while __l    (1分)______________________________________________________:

        f, l = _f(l[0])    (1分)_______________________, _l[1:]    (1分)_______________________
```

Problem 5: Composite Functions

```
def composite_setsuna(n):
    """Composites N functions.
    >>> composite_setsuna(3)(lambda x: x + 2)(lambda x: x * 2)(lambda x: x - 2)(5)
    8
    >>> composite_setsuna(2)(lambda x: x ** 2)(lambda x: x // 2)(9)
    16
    """
    func = lambda x: x
    def helper(f):
        if n < 0:
            return func(f)
        n -= 1
        func = lambda x: func(f(x))
        return helper
    return helper
```

当n变为0时，没有更多函数，应该进行函数调用

注意：变量f可以是函数，也可以是数值！
所以return func(f)
等价于return (lambda x: func(x))(f)
但是不等价于return lambda x: func(x)

Problem 5: Composite Functions

```python
def composite_setsuna(n):
    """Composites N functions.
    >>> composite_setsuna(3)(lambda x: x + 2)(lambda x: x * 2)(lambda x: x - 2)(5)
    8
    >>> composite_setsuna(2)(lambda x: x ** 2)(lambda x: x // 2)(9)
    16
    """
    func = lambda x: x
    def helper(f):
        if n <= 0:
            return func(f)
        n -= 1
        func = lambda x: func(f(x))
        return helper
    return helper
```

Reference before assignment

Problem 5: Composite Functions

```
def composite_setsuna(n):
    """Composites N functions.
    >>> composite_setsuna(3)(lambda x: x + 2)(lambda x: x * 2)(lambda x: x - 2)(5)
    8
    >>> composite_setsuna(2)(lambda x: x ** 2)(lambda x: x // 2)(9)
    16
    """
    func = lambda x: x
    def helper(f):
        if n <= 0:
            return func(f)
        n -= 1
        func = lambda x: func(f(x))
        return helper
    return helper
```

```
nonlocal n, func
```

Infinite recursion

```
func = lambda x: func(...)
```

```
func(func(func(func(func(func(func(func(func...
```

Problem 5: Composite Functions

```
def composite_setsuna(n):
    """Composites N functions.
    >>> composite_setsuna(3)(lambda x: x + 2)(lambda x: x * 2)(lambda x: x - 2)(5)
    8
    >>> composite_setsuna(2)(lambda x: x ** 2)(lambda x: x // 2)(9)
    16
    """
    func = lambda x: x
    def helper(f):              nonlocal n, func
        if n <= 0:
            return func(f)
        n -= 1
        func =  (lambda g: lambda x: g(f(x)))(func)
        return helper
    return helper
```

由于nonlocal的存在，调用helper就会改变func所指向的函数，这是一种副作用

因此在第5.2、5.3题中，只需要调用func就可以了，不需要取返回值！
这是两个comp的差别之一

Problem 5: Composite Functions

```python
def composite_setsuna(n):
    """Composites N functions.
    >>> composite_setsuna(3)(lambda x: x + 2)(lambda x: x * 2)(lambda x: x - 2)(5)
    8
    >>> composite_setsuna(2)(lambda x: x ** 2)(lambda x: x // 2)(9)
    16
    """
    func = lambda x: x
    def helper(f):
        if n <= 0:
            return func(f)
        n -= 1
        func = (lambda g: lambda x: g(f(x)))(func)
        return helper
    return helper
```

nonlocal n, func

当n<=0时，函数可以“正常”处理

Problem 5: Composite Functions

```
def composite_kazusa(n):
    if n == 1:
        return lambda f: lambda x: f(x)
    return lambda f: lambda g: composite_kazusa(n - 1)(lambda x: f(g(x)))
```

当n<=0时，函数不能“正常”处理

5.3的一种简单做法就是发现这两个函数对**非法输入**的处理是不同的，一个会返回函数，一个会返回新的comp

Problem 6: Trees

前两问都是
送！分！题！

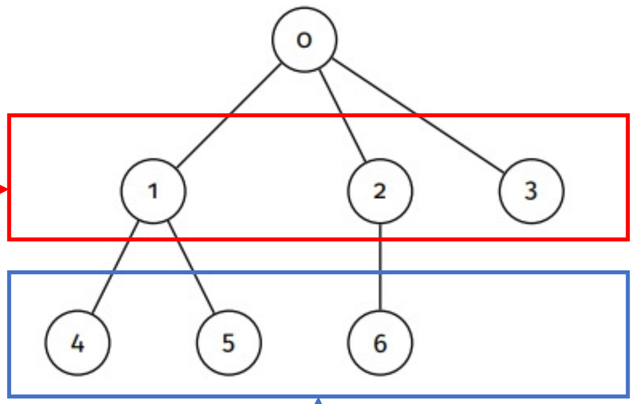
第二问中，`all([]) == True`，因此代码显然不对
这道题的要给出代码出错的例子，`t1`、`t2`必须是合法的

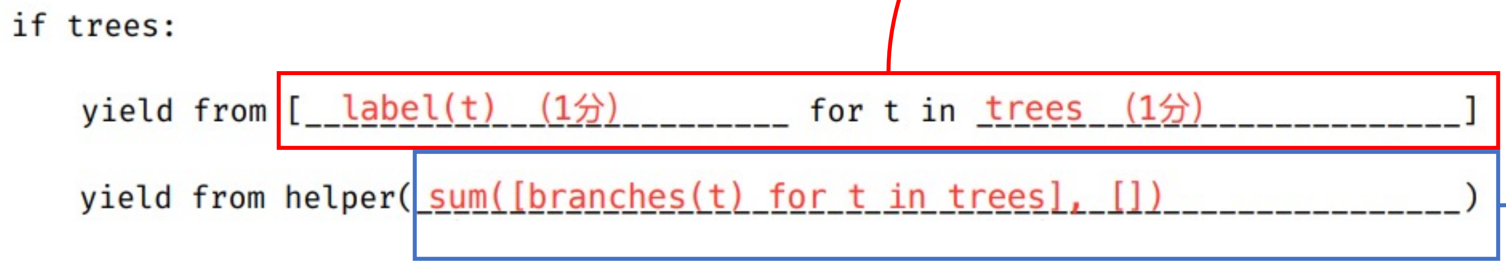
```
t1 = tree(0)
t2 = tree(0, [tree(0)])
```

Problem 6: Trees

第三问：

- trees：很多个tree
- 每次调用helper都是输出一行





```
if trees:
    yield from [__label(t)__(1分)__________ for t in _trees__(1分)_______________]
    yield from helper(_sum([branches(t) for t in trees], [])_______________)
```

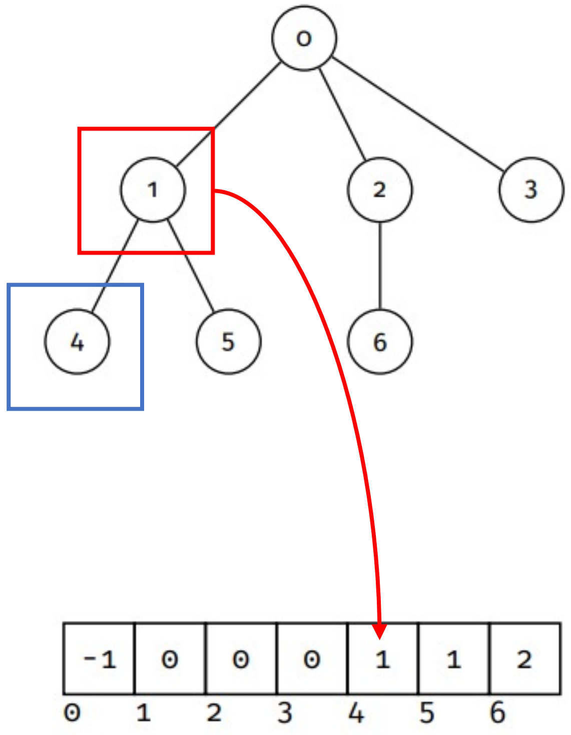
Problem 6: Trees

第四问：真的好基础的递归

不应该放最后一题，大家都没时间做

每空1分

```
result = [-1 for i in range(n)]

def helper(tree, parent_label):

    result[label(tree)]_________________________________ = parent_label

    for branch in branches(tree)____________________________________:

        helper(branch___________________, label(tree)_______________)

helper(t, -1_______________)

return result
```



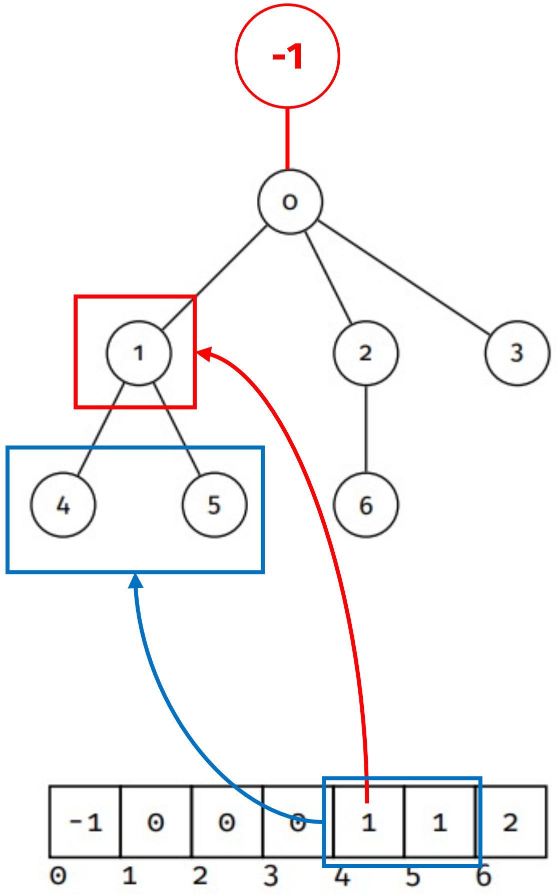
Problem 6: Trees

第四问：真的好基础的递归

不应该放最后一题，大家都没时间做

每空1分

```
def helper(current_label):
    return tree(_current_label________, [_helper(index)_____ for index \
                in __range(len(lst))__ if __lst[index] == current_label__])
return branches(_helper(-1)__________)[0]
```

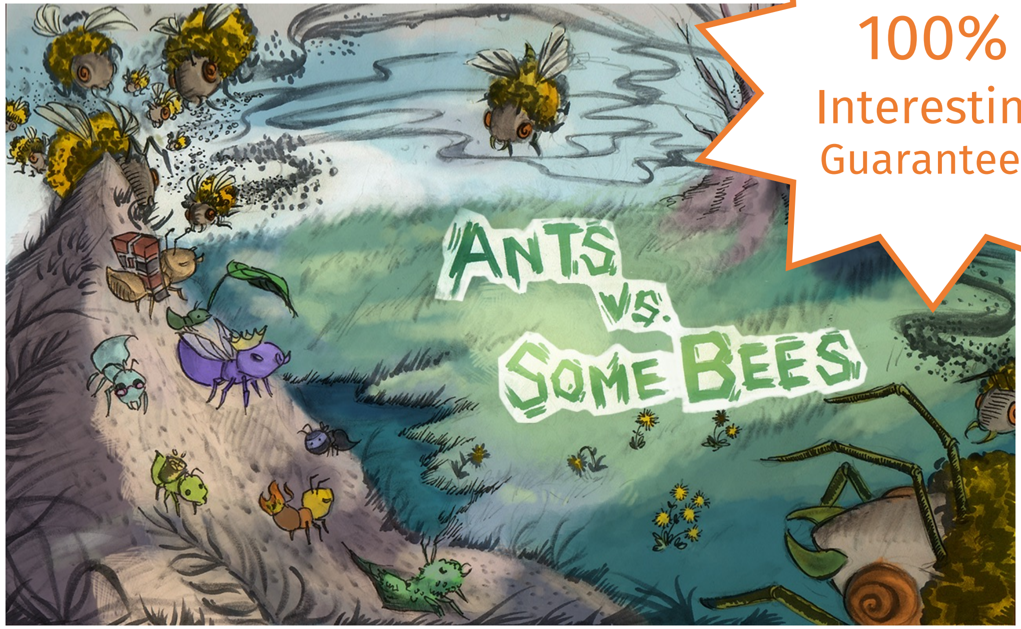


This week, let's learn OOP

Hw06: OOP problems

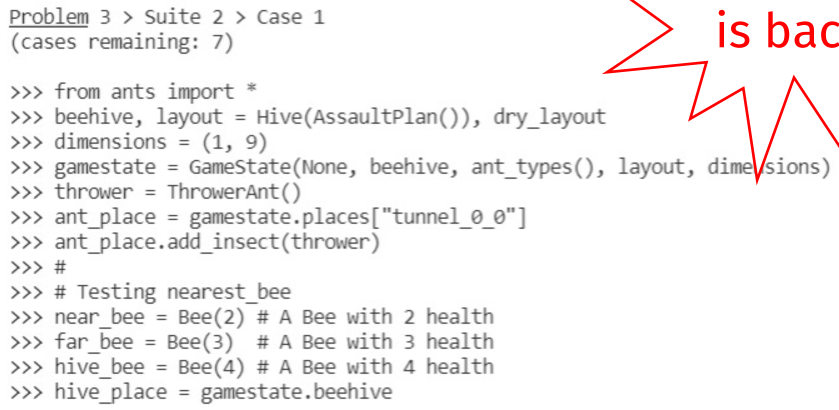
Proj03: Ants vs. Some Bees

亲，记得先认真学习微积分和线代



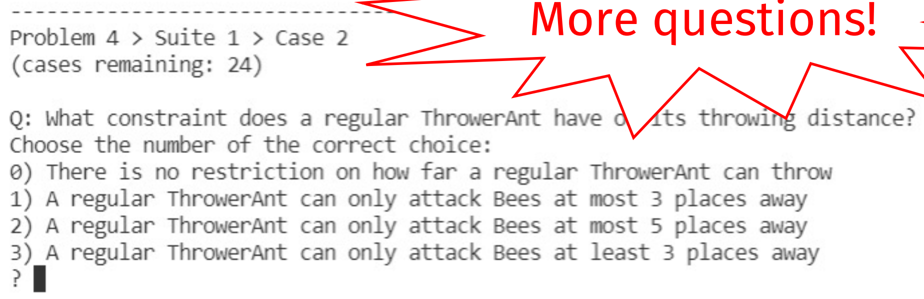
100%
Interesting
Guaranteed

WWPD
is back!



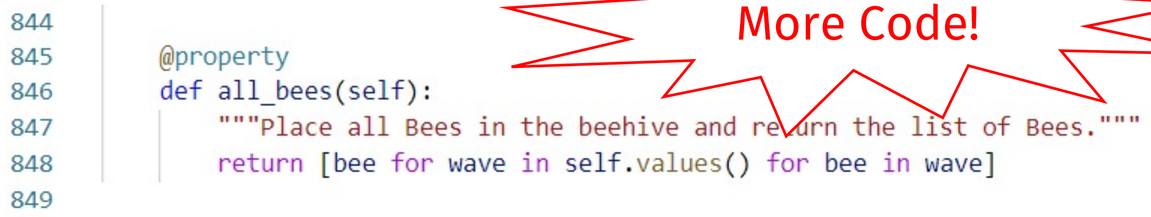
```
Problem 3 > Suite 2 > Case 1
(cases remaining: 7)

>>> from ants import *
>>> beehive, layout = Hive(AssaultPlan()), dry_layout
>>> dimensions = (1, 9)
>>> gamestate = GameState(None, beehive, ant_types(), layout, dimensions)
>>> thrower = ThrowerAnt()
>>> ant_place = gamestate.places["tunnel_0_0"]
>>> ant_place.add_insect(thrower)
>>> #
>>> # Testing nearest_bee
>>> near_bee = Bee(2) # A Bee with 2 health
>>> far_bee = Bee(3)  # A Bee with 3 health
>>> hive_bee = Bee(4) # A Bee with 4 health
>>> hive_place = gamestate.beehive
```

More questions!



```
---------------------------------
Problem 4 > Suite 1 > Case 2
(cases remaining: 24)

Q: What constraint does a regular ThrowerAnt have on its throwing distance?
Choose the number of the correct choice:
0) There is no restriction on how far a regular ThrowerAnt can throw
1) A regular ThrowerAnt can only attack Bees at most 3 places away
2) A regular ThrowerAnt can only attack Bees at most 5 places away
3) A regular ThrowerAnt can only attack Bees at least 3 places away
?
```

More Code!



```
@property
def all_bees(self):
    """Place all Bees in the beehive and return the list of Bees."""
    return [bee for wave in self.values() for bee in wave]
```