Discussion 1

While and If

Q1:Race

以下代码片段有问题，在某些情况下会导致输出错误或者是死循环

> 题目陈述：
> 定义函数 `race(x,y)` ，该函数输入为 🐢 乌龟(tortoise)的速度 x 和🐰 兔子(hare)的速度 y
> 🐢 乌龟每分钟走 x 米，🐰 兔子每分钟走 y 米，但是🐰 兔子每走五分钟就睡五分钟
>
> 数据范围：
> $x < y < 2x$
>
> 输入输出：
> >>> race(5, 7)
> 7
> >>> race(2, 4)
> 10

```
def race(x, y):

"""The tortoise always walks x feet per minute, while the hare repeatedly
runs y feet per minute for 5 minutes, then rests for 5 minutes. Return how
many minutes pass until the tortoise first catches up to the hare.

>>> race(5, 7) # After 7 minutes, both have gone 35 steps
7

>>> race(2, 4) # After 10 minutes, both have gone 20 steps
10

"""

    assert y > x and y <= 2 * x, 'the hare must be fast but not too fast'
    tortoise, hare, minutes = 0, 0, 0

    while minutes == 0 or tortoise - hare:
        tortoise += x
        if minutes % 10 < 5:
            hare += y
        minutes += 1

    return minutes
```

1) 找满足条件的两个正整数 x, y 使得函数 `race(x,y)` 返回值错误。

2) 找满足条件的两个正整数 x, y 使得函数 `race(x,y)` 陷入死循环。

3) 为什么该函数错误？如何更改可以使该函数返回正确的值？

答案：

1)

$$x = 4, y = 6 \Rightarrow minutes = 15$$

2)

$$x = 3, y = 4 \Rightarrow minutes = \infty$$

3)

因为

```
while minutes == 0 or tortoise - hare:
```

该语句中`while`循环的条件是错的，该循环意为：

“如果 时间为零 和 乌龟跑的路程和兔子不相等 两个条件中有一个是真，该循环就会进行”

该程序应该改为：

```
def race(x, y):
    assert y > x and y <= 2 * x, 'the hare must be fast but not too fast'
    return 5 * y / x
```

或者

```
def race(x, y):

    assert y > x and y <= 2 * x, 'the hare must be fast but not too fast'
    tortoise, hare, minutes = 0, 0, 0

    while tortoise < hare or minutes == 0:
        tortoise += x
        if minutes % 10 < 5:
            hare += y
        minutes += 1

    return minutes
```

Q2:Fizzbuzz

问题陈述：
经典的Fizzbuzz序列问题，写一个函数 `fizzbuzz(n)`
函数 `fizzbuzz(n)` 接受一个输入 n 输出从 1 到 n 的整数
但是对于每一个 i：
如果 $i(\mod 3) = 0, i(\mod 5) \neq 0$ 的时候输出 `fizz`
如果 $i(\mod 3) \neq 0, i(\mod 5) = 0$ 的时候输出 `buzz`
如果 $i(\mod 3) = 0, i(\mod 5) = 0$ 的时候输出 `fizzbuzz`

数据范围：
$n \in \mathbb{N}_+$

输入输出：
>>> result = fizzbuzz(16)
1
2
fizz
4
buzz
fizz
7
8
9
buzz
11
fizz
13
14
fizzbuzz
16
>>> print(result)
None

答案：

```
def fizzbuzz(n):
    """
    >>> result = fizzbuzz(16)
    1
    2
    fizz
    4
    buzz
    fizz
    7
    8
    fizz
```

```
    buzz
    11
    fizz
    13
    14
    fizzbuzz
    16
    >>> print(result)
    None
    """
    count = 1
    while count <= n:
        if   count % 3 == 0 and count % 5 != 0:
            print("fizz")
        elif count % 3 != 0 and count % 5 == 0:
            print("buzz")
        elif count % 15 == 0:
            print("fizzbuzz")
        else:
            print(count)
        count += 1
```

Problem Solving

Q3:Is Prime?

> 问题陈述：
> 写一个函数 `is_prime(n)` 来判断输入的数字 n 是不是质数
>
> 数据范围：
> $n \in \mathbb{N}_+$
>
> 输入输出：
> >>> is_prime(10)
> False
> >>> is_prime(7)
> True
> >>> is_prime(1)
> False

答案：

```
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
```

```
    >>> is_prime(1) # one is not a prime number!!
    False
    """
    factor = 2
    if n <= 1:
        return False
    while factor < n:
        if n % factor == 0:
            return False
        factor += 1
    return True
```

Q4:Unique Digits

> 问题陈述：
> 写一个函数 `unique_digits(n)` 判断正整数 n 的数码中有多少不一样的数字。
>
> 数据范围：
> $n \in \mathbb{N}_+$
>
> 输入输出：
> >>> unique_digits(8675309) # All are unique
> 7
> >>> unique_digits(13173131) # 1, 3, and 7
> 3
> >>> unique_digits(101) # 0 and 1
> 2
>
> **Hint:**
> 如果另外加上一个函数 `has_digit(n,k)` 如下：

答案：

```
def unique_digits(n):
    """
    Return the number of unique digits in positive integer n.
    >>> unique_digits(8675309) # All are unique
    7
    >>> unique_digits(13173131) # 1, 3, and 7
    3
    >>> unique_digits(101) # 0 and 1
    2
    """
    factor = 0
    count  = 0
    while factor < 10:
        if has_digit(n, factor):
            count += 1
```

```
            factor += 1
        return count

    def has_digit(n, k):
        """
        Returns whether k is a digit in n.
        >>> has_digit(10, 1)
        True
        >>> has_digit(12, 7)
        >>> False
        """
        assert k >= 0 and k < 10
        while n >= 0:
            if n % 10 == k:
                return True
            elif n < 10 and n != k:
                break
            n = n // 10
        return False
```

Environment Diagrams

Q5:Bottles

1.在环境图中，什么决定帧的数量?

> a) 代码中定义的函数的数量
> b) 代码中调用表达式的数量
> c) 代码中 `return` 语句的数量
> d) 在代码运行时，用户定义的函数被调用的次数

2.当返回值 `pass_it(bottles)` 发生了什么?

> a) 它的值在全局帧中被赋给 `remaining`
> b) 它的值在全局帧中被赋给 `bottles`
> c) 它的值在全局帧中被赋给 `pass_it`
> d) 以上均不正确

3. `bottles = 98` 这行代码对全局帧有什么影响?

> a) 暂时改变了全局帧中 `bottles` 的值
> b) 永久改变了全局帧中 `bottles` 的值
> c) 没有对全局帧产生影响

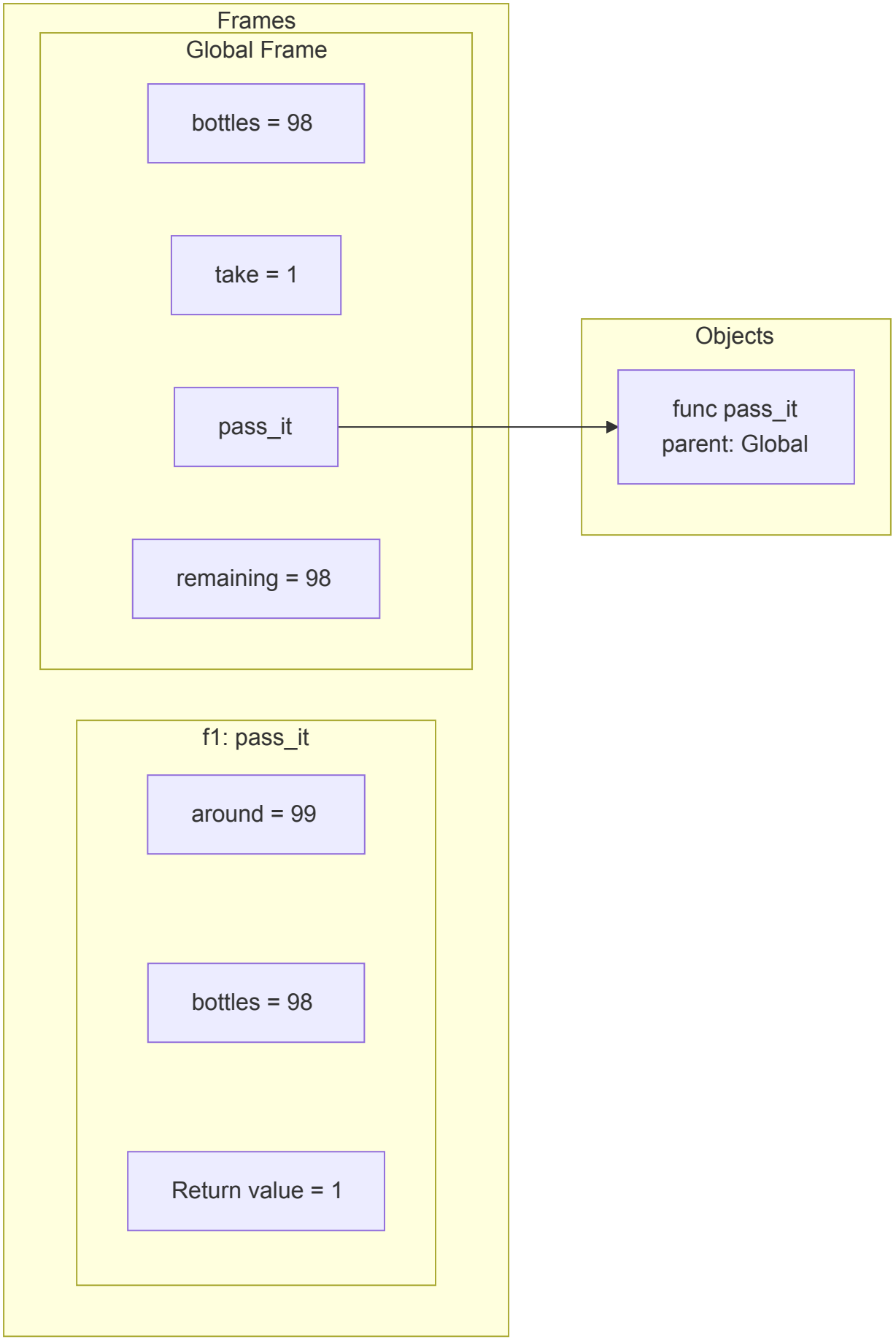
答案：

1.d 2.d 3.c

```
bottles = 99
take    = 1

def pass_it(around):
    bottles = 98
    return take

remaining = bottles - pass_it(bottles)
bottles   = remaining
```



Q6:Double Trouble

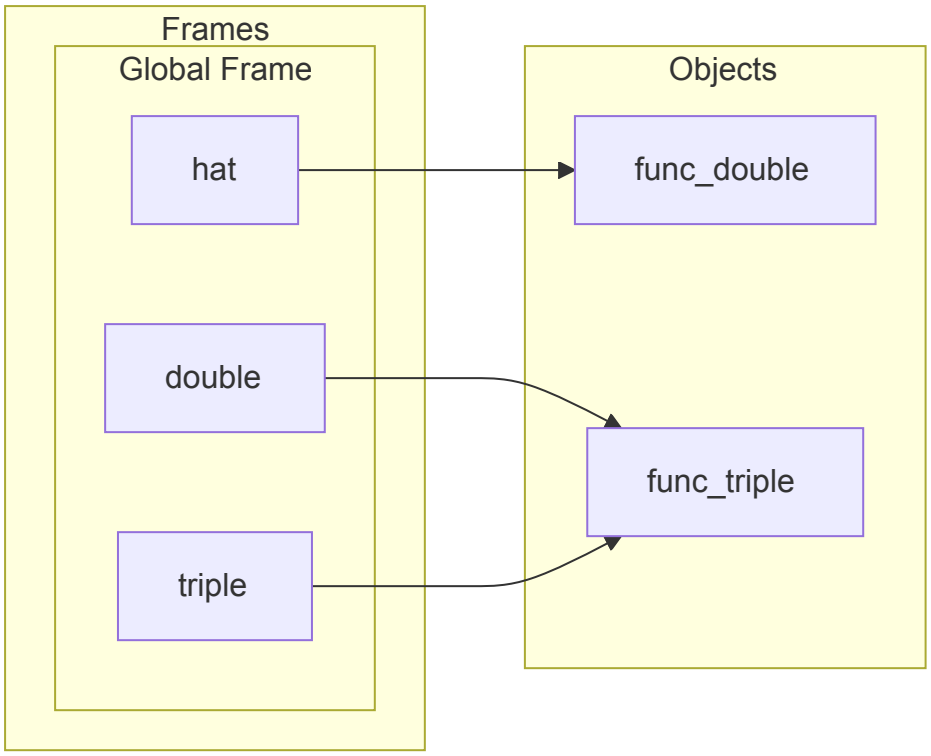
为以下代码画出环境图：

```
def double(x):
    return x * 2

def triple(x):
    return x * 3

hat = double
double = triple
```

答案：



Optional:Exam Practice

Q7:Repeating

> 问题陈述：
> 函数 `repeating(t, n)` 接受两个正整数输入，判断正整数 n 是不是由 t 个数码组成的序列
> 只需要填空。
>
> 数据范围：
> $t, n \in \mathbb{N}_+$
>
> 输入输出：
> >>> repeating(1, 6161)
> False
> >>> repeating(2, 6161) # repeats 61 (2 digits)
> True

>>> repeating(3, 6161)
False
>>> repeating(4, 6161) # repeats 6161 (4 digits)
True
>>> repeating(5, 6161) # there are only 4 digits
False

```
def repeating(t, n):

    """Return whether t digits repeat to form positive integer n.

    >>> repeating(1, 6161)
    False
    >>> repeating(2, 6161) # repeats 61 (2 digits)
    True
    >>> repeating(3, 6161)
    False
    >>> repeating(4, 6161) # repeats 6161 (4 digits)
    True
    >>> repeating(5, 6161) # there are only 4 digits
    False
    """

    if pow(10, t-1) > n: # make sure n has at least t digits
        return False
    end = n % pow(10,t)
    rest = n
    while rest:
        if rest % pow(10, t) != end:
            return False
        rest = rest // pow(10, t)
    return True
```