Midterm

You Zhou

San Francisco Bay University

CE 450

July 1st, 2023

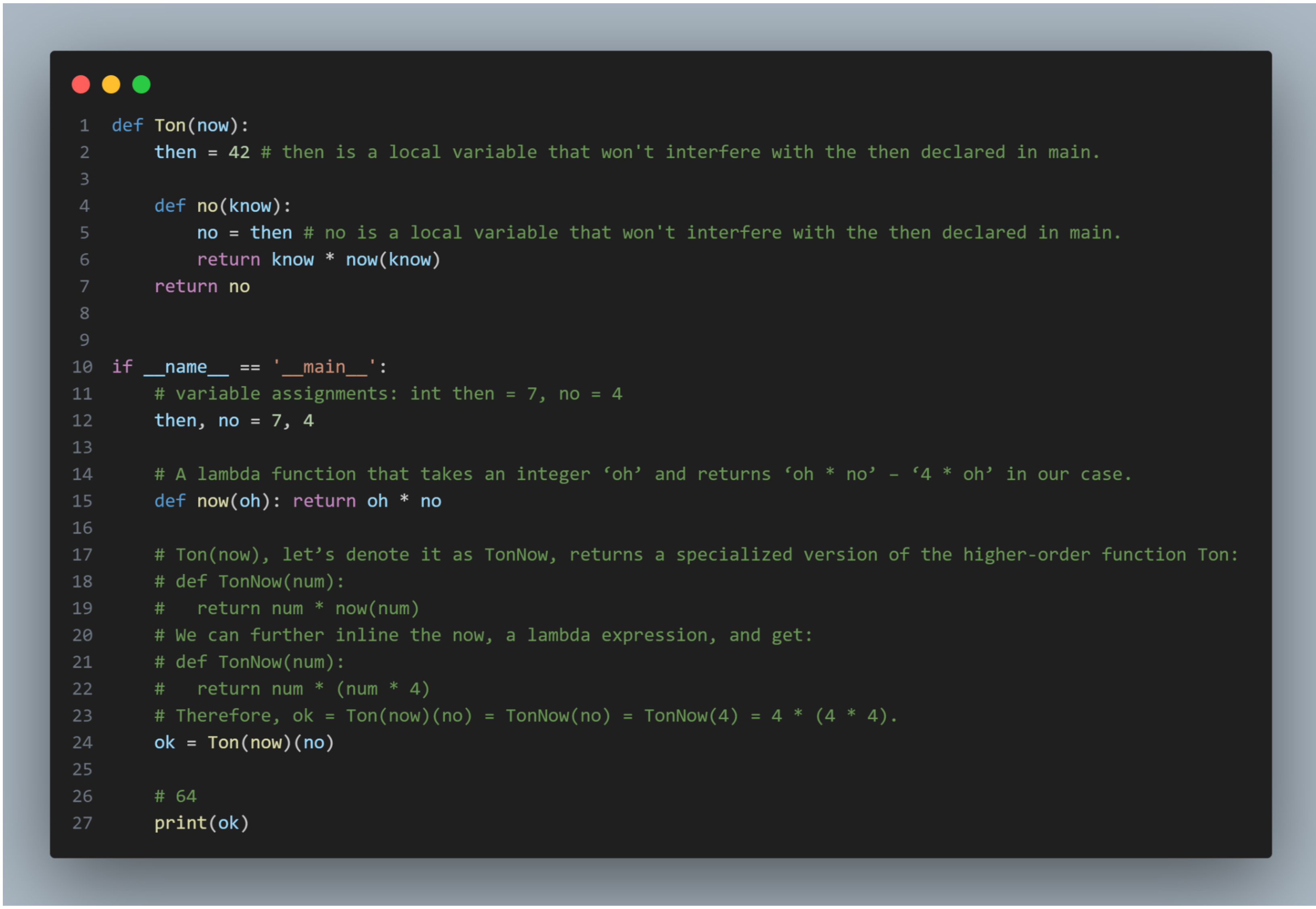
Question 1.1

1. Suitable for a single-chip system.
2. Capable of real-time processing.
3. As power-efficient as possible.

Question 1.2

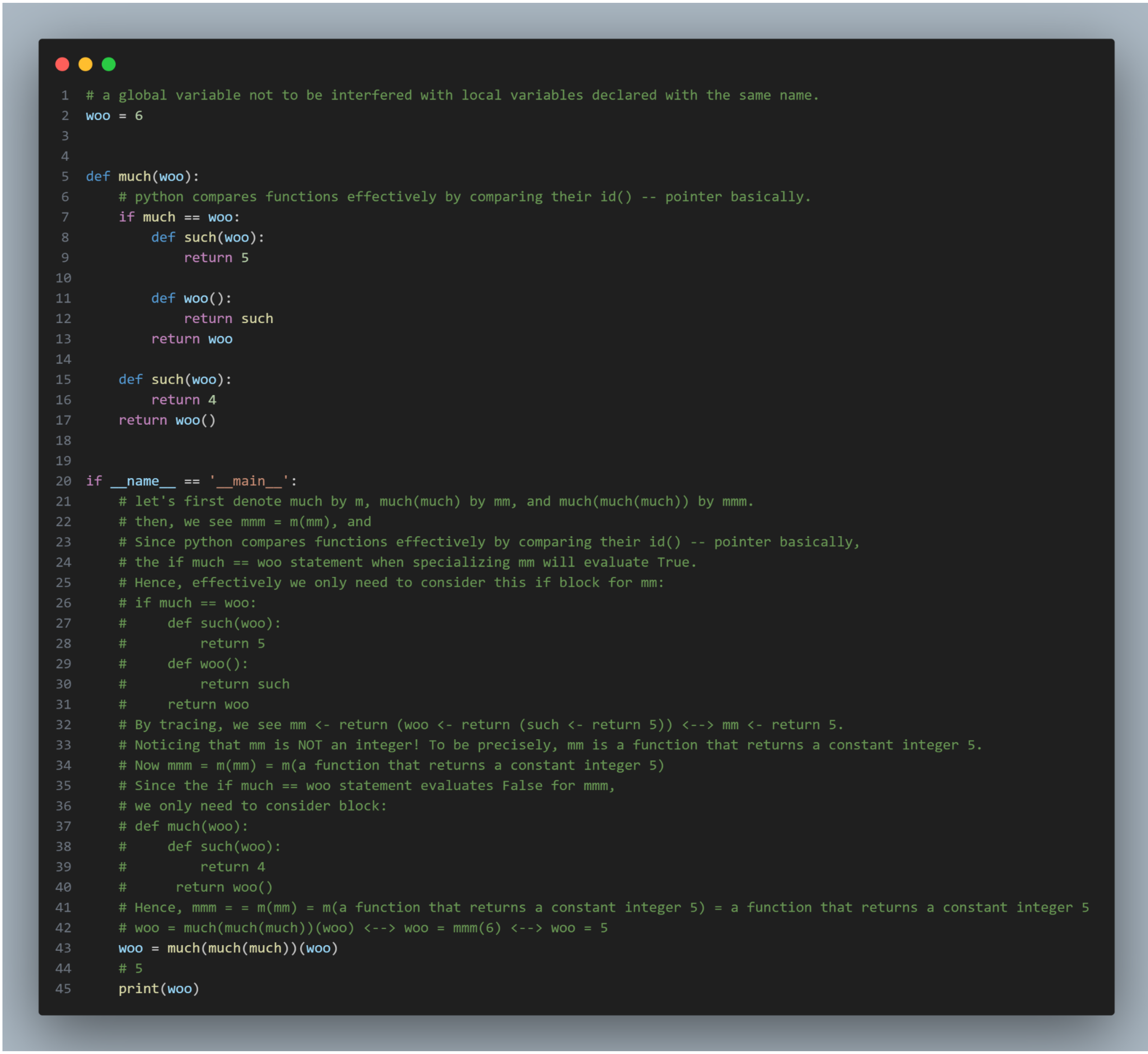
A family of RISC architectures that preset a set of common facilities for the interactions between software and hardware.

Question 2



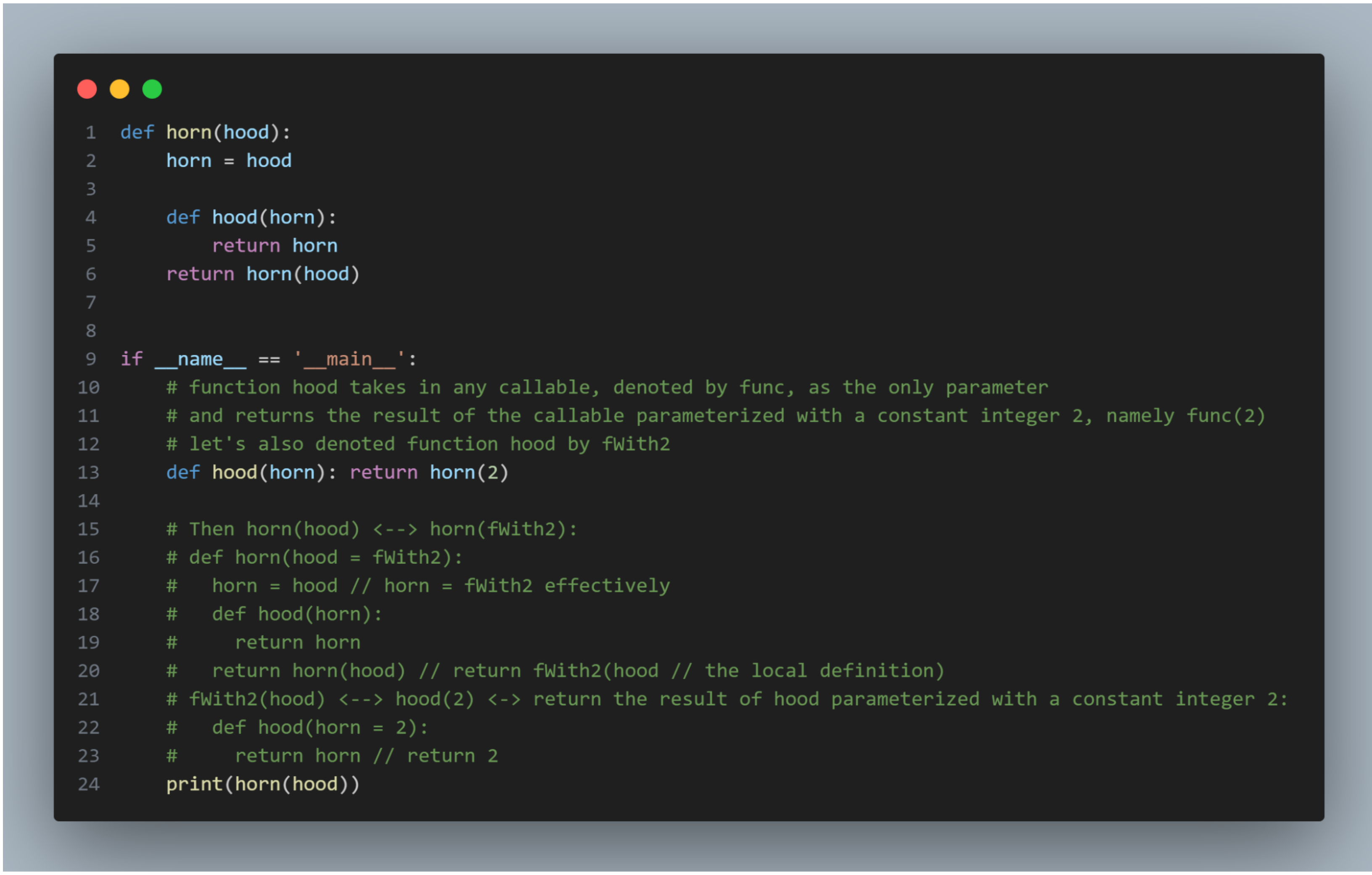
```python
def Ton(now):
    then = 42 # then is a local variable that won't interfere with the then declared in main.

    def no(know):
        no = then # no is a local variable that won't interfere with the then declared in main.
        return know * now(know)
    return no


if __name__ == '__main__':
    # variable assignments: int then = 7, no = 4
    then, no = 7, 4

    # A lambda function that takes an integer ‘oh’ and returns ‘oh * no’ – ‘4 * oh’ in our case.
    def now(oh): return oh * no

    # Ton(now), let’s denote it as TonNow, returns a specialized version of the higher-order function Ton:
    # def TonNow(num):
    #   return num * now(num)
    # We can further inline the now, a lambda expression, and get:
    # def TonNow(num):
    #   return num * (num * 4)
    # Therefore, ok = Ton(now)(no) = TonNow(no) = TonNow(4) = 4 * (4 * 4).
    ok = Ton(now)(no)

    # 64
    print(ok)
```

Question 3



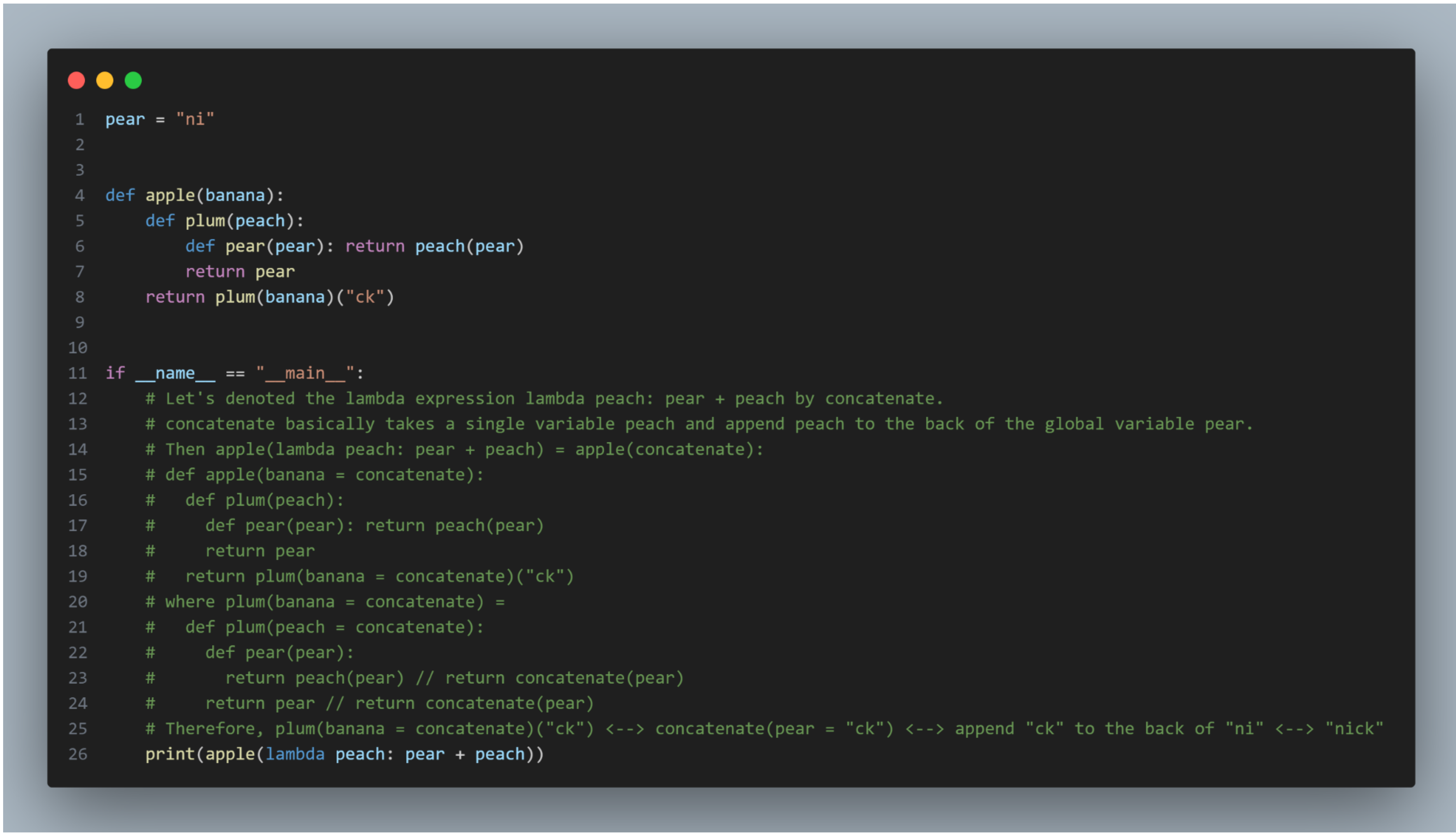
```python
# a global variable not to be interfered with local variables declared with the same name.
woo = 6


def much(woo):
    # python compares functions effectively by comparing their id() -- pointer basically.
    if much == woo:
        def such(woo):
            return 5

        def woo():
            return such
        return woo

    def such(woo):
        return 4
    return woo()


if __name__ == '__main__':
    # let's first denote much by m, much(much) by mm, and much(much(much)) by mmm.
    # then, we see mmm = m(mm), and
    # Since python compares functions effectively by comparing their id() -- pointer basically,
    # the if much == woo statement when specializing mm will evaluate True.
    # Hence, effectively we only need to consider this if block for mm:
    # if much == woo:
    #     def such(woo):
    #         return 5
    #     def woo():
    #         return such
    #     return woo
    # By tracing, we see mm <- return (woo <- return (such <- return 5)) <--> mm <- return 5.
    # Noticing that mm is NOT an integer! To be precisely, mm is a function that returns a constant integer 5.
    # Now mmm = m(mm) = m(a function that returns a constant integer 5)
    # Since the if much == woo statement evaluates False for mmm,
    # we only need to consider block:
    # def much(woo):
    #     def such(woo):
    #         return 4
    #      return woo()
    # Hence, mmm = = m(mm) = m(a function that returns a constant integer 5) = a function that returns a constant integer 5
    # woo = much(much(much))(woo) <--> woo = mmm(6) <--> woo = 5
    woo = much(much(much))(woo)
    # 5
    print(woo)
```

Question 4



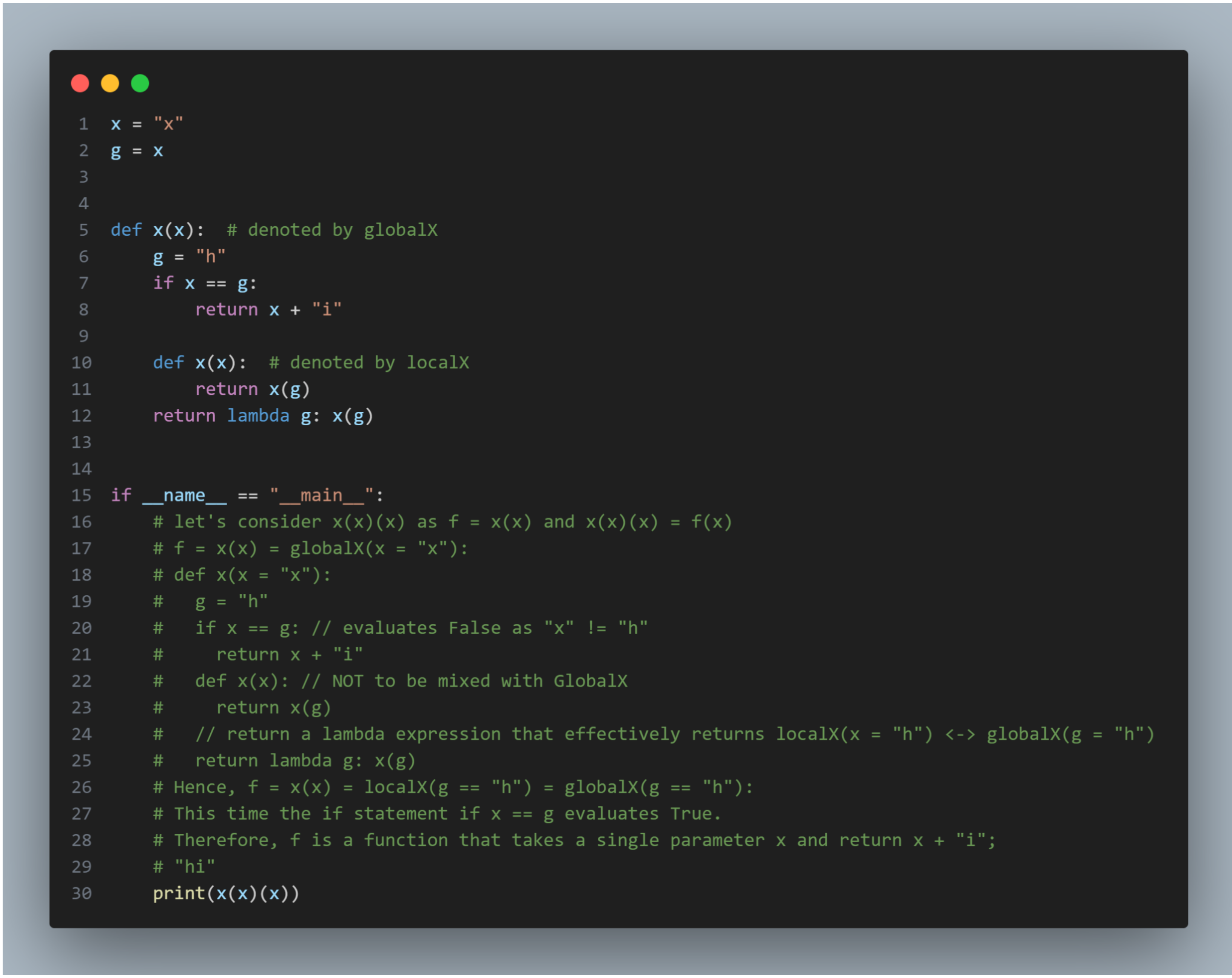
```python
def horn(hood):
    horn = hood

    def hood(horn):
        return horn
    return horn(hood)


if __name__ == '__main__':
    # function hood takes in any callable, denoted by func, as the only parameter
    # and returns the result of the callable parameterized with a constant integer 2, namely func(2)
    # let's also denoted function hood by fWith2
    def hood(horn): return horn(2)

    # Then horn(hood) <--> horn(fWith2):
    # def horn(hood = fWith2):
    #   horn = hood // horn = fWith2 effectively
    #   def hood(horn):
    #     return horn
    #   return horn(hood) // return fWith2(hood // the local definition)
    # fWith2(hood) <--> hood(2) <-> return the result of hood parameterized with a constant integer 2:
    #   def hood(horn = 2):
    #     return horn // return 2
    print(horn(hood))
```

Question 5



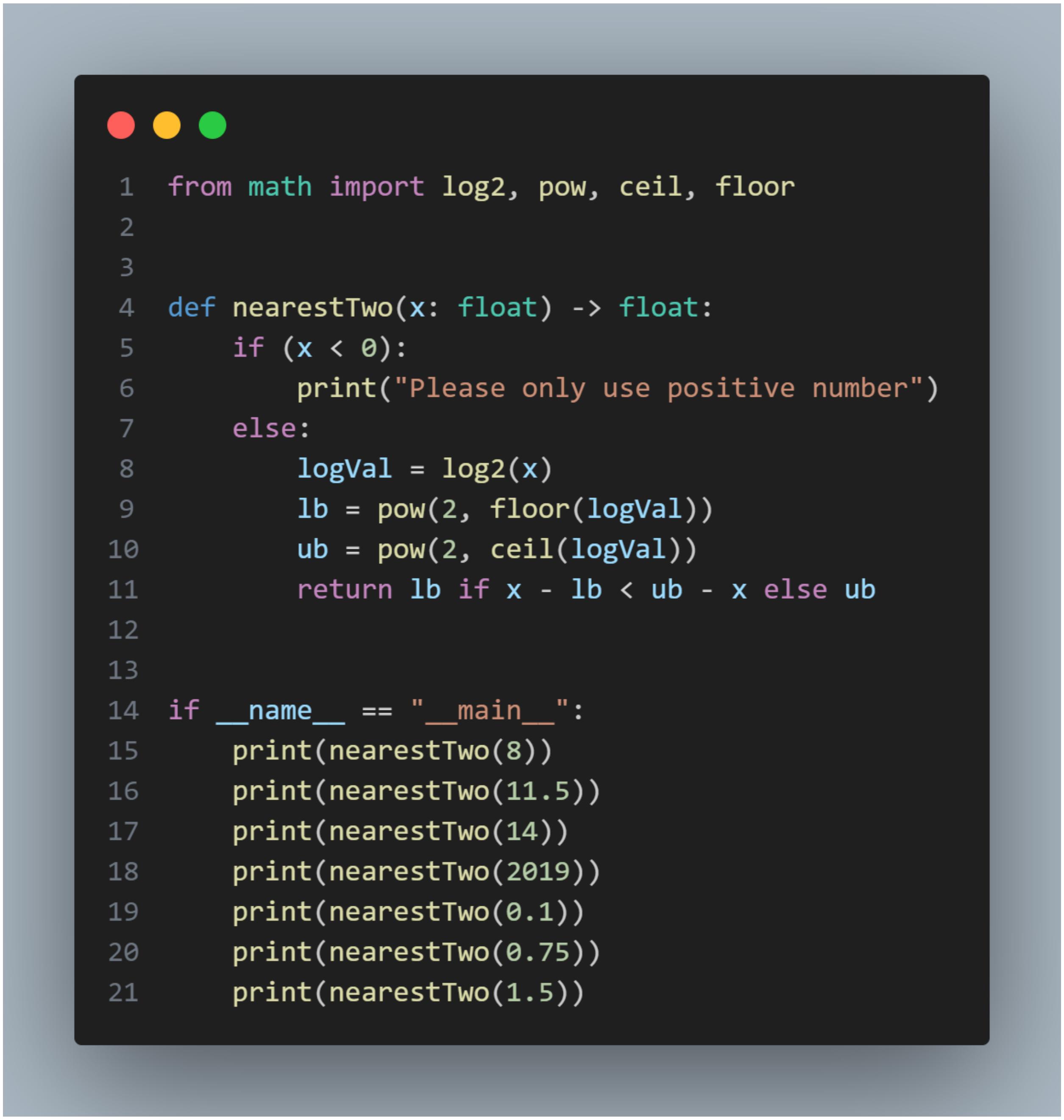
```python
pear = "ni"


def apple(banana):
    def plum(peach):
        def pear(pear): return peach(pear)
        return pear
    return plum(banana)("ck")


if __name__ == "__main__":
    # Let's denoted the lambda expression lambda peach: pear + peach by concatenate.
    # concatenate basically takes a single variable peach and append peach to the back of the global variable pear.
    # Then apple(lambda peach: pear + peach) = apple(concatenate):
    # def apple(banana = concatenate):
    #   def plum(peach):
    #     def pear(pear): return peach(pear)
    #     return pear
    #   return plum(banana = concatenate)("ck")
    # where plum(banana = concatenate) =
    #   def plum(peach = concatenate):
    #     def pear(pear):
    #       return peach(pear) // return concatenate(pear)
    #     return pear // return concatenate(pear)
    # Therefore, plum(banana = concatenate)("ck") <--> concatenate(pear = "ck") <--> append "ck" to the back of "ni" <--> "nick"
    print(apple(lambda peach: pear + peach))
```

Question 6



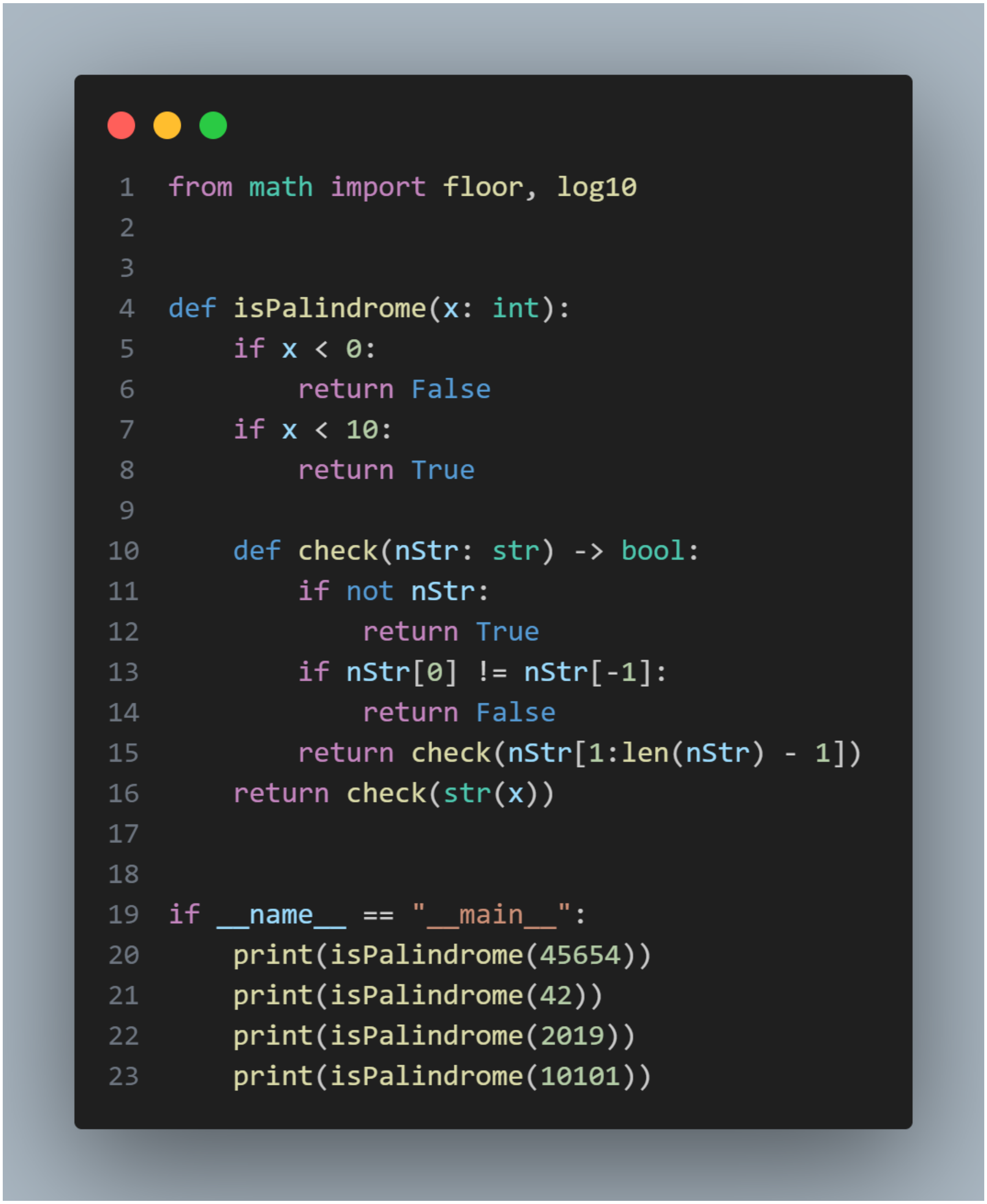
```python
x = "x"
g = x


def x(x):  # denoted by globalX
    g = "h"
    if x == g:
        return x + "i"

    def x(x):  # denoted by localX
        return x(g)
    return lambda g: x(g)


if __name__ == "__main__":
    # let's consider x(x)(x) as f = x(x) and x(x)(x) = f(x)
    # f = x(x) = globalX(x = "x"):
    # def x(x = "x"):
    #   g = "h"
    #   if x == g: // evaluates False as "x" != "h"
    #     return x + "i"
    #   def x(x): // NOT to be mixed with GlobalX
    #     return x(g)
    #   // return a lambda expression that effectively returns localX(x = "h") <-> globalX(g = "h")
    #   return lambda g: x(g)
    # Hence, f = x(x) = localX(g == "h") = globalX(g == "h"):
    # This time the if statement if x == g evaluates True.
    # Therefore, f is a function that takes a single parameter x and return x + "i";
    # "hi"
    print(x(x)(x))
```

Question 7



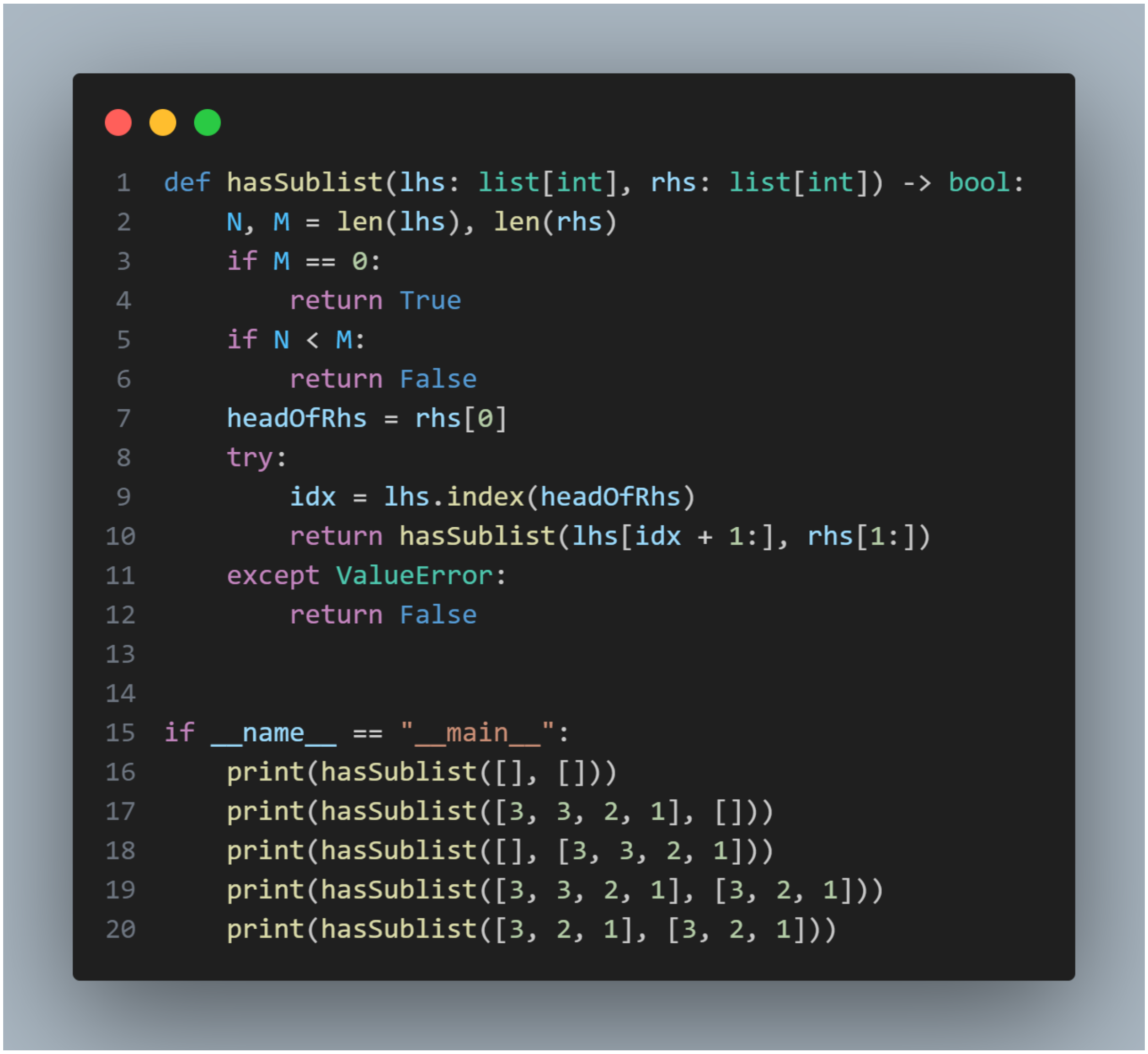
```python
from math import log2, pow, ceil, floor


def nearestTwo(x: float) -> float:
    if (x < 0):
        print("Please only use positive number")
    else:
        logVal = log2(x)
        lb = pow(2, floor(logVal))
        ub = pow(2, ceil(logVal))
        return lb if x - lb < ub - x else ub


if __name__ == "__main__":
    print(nearestTwo(8))
    print(nearestTwo(11.5))
    print(nearestTwo(14))
    print(nearestTwo(2019))
    print(nearestTwo(0.1))
    print(nearestTwo(0.75))
    print(nearestTwo(1.5))
```

Question 8



```python
from math import floor, log10


def isPalindrome(x: int):
    if x < 0:
        return False
    if x < 10:
        return True

    def check(nStr: str) -> bool:
        if not nStr:
            return True
        if nStr[0] != nStr[-1]:
            return False
        return check(nStr[1:len(nStr) - 1])
    return check(str(x))


if __name__ == "__main__":
    print(isPalindrome(45654))
    print(isPalindrome(42))
    print(isPalindrome(2019))
    print(isPalindrome(10101))
```

Question 9



```python
def hasSublist(lhs: list[int], rhs: list[int]) -> bool:
    N, M = len(lhs), len(rhs)
    if M == 0:
        return True
    if N < M:
        return False
    headOfRhs = rhs[0]
    try:
        idx = lhs.index(headOfRhs)
        return hasSublist(lhs[idx + 1:], rhs[1:])
    except ValueError:
        return False


if __name__ == "__main__":
    print(hasSublist([], []))
    print(hasSublist([3, 3, 2, 1], []))
    print(hasSublist([], [3, 3, 2, 1]))
    print(hasSublist([3, 3, 2, 1], [3, 2, 1]))
    print(hasSublist([3, 2, 1], [3, 2, 1]))
```