15-110 Principles of Computing – F21

Lecture 10:

Debugging, Loops

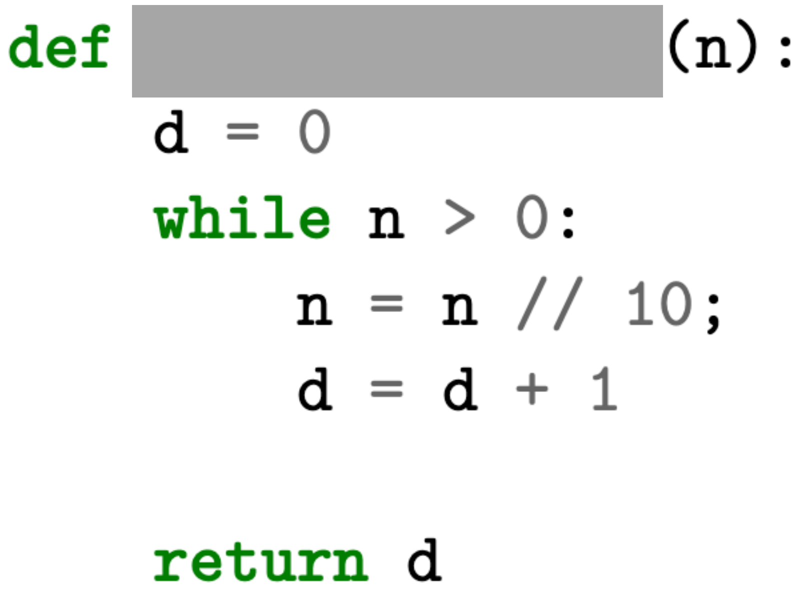
Teacher:

Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر

Carnegie Mellon University Qatar

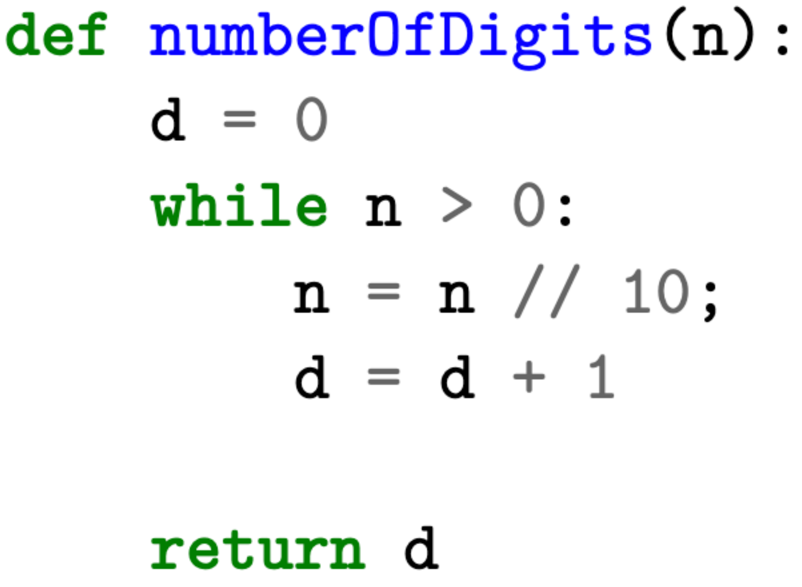
What the code does?



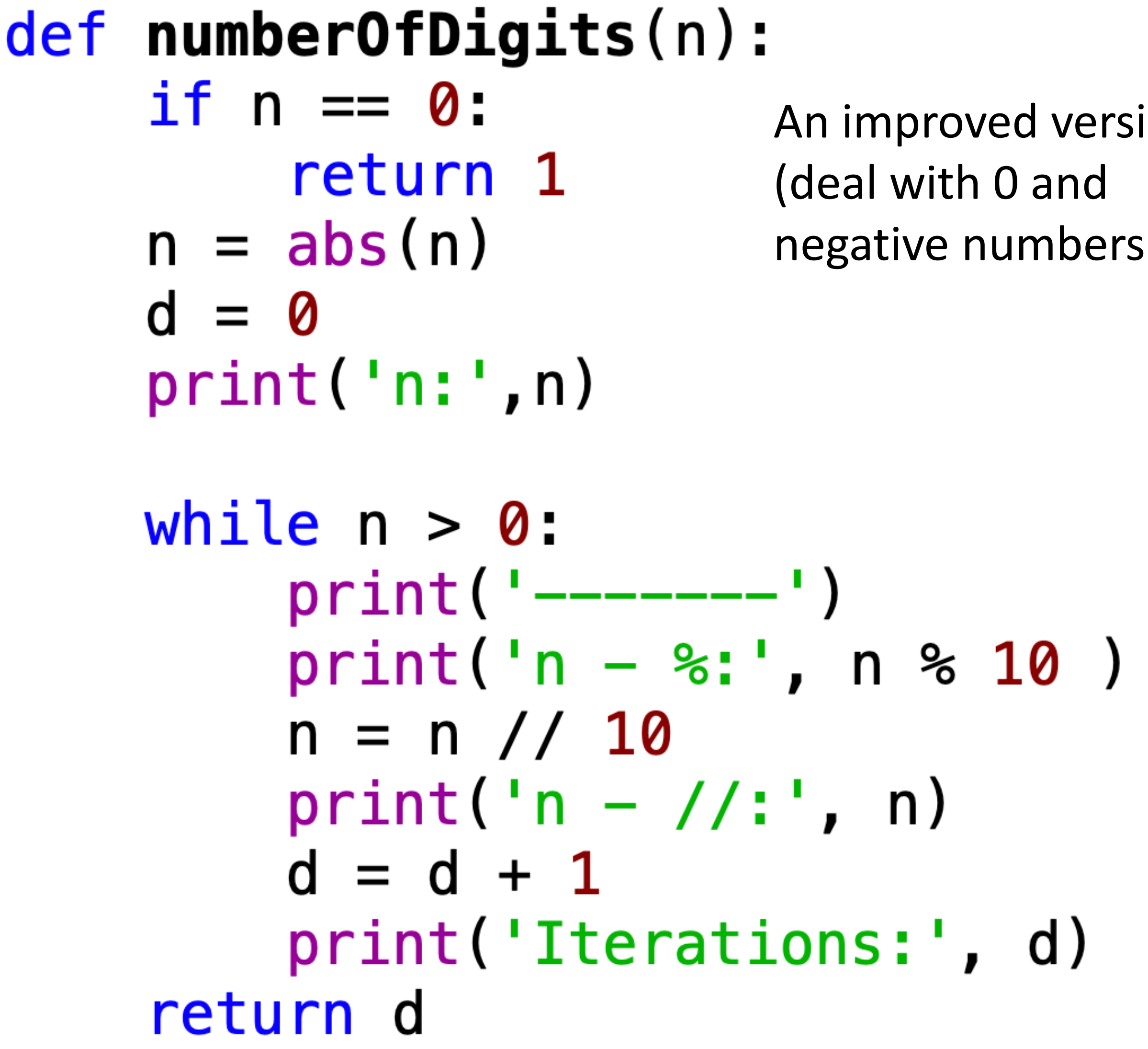
```
def [redacted](n):
    d = 0
    while n > 0:
        n = n // 10;
        d = d + 1

    return d
```

Let’s use `print()` to gain insights in the code!

What the code does? Use `print()` to get a better understanding!



```
def numberOfDigits(n):
    d = 0
    while n > 0:
        n = n // 10;
        d = d + 1

    return d
```



```
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    d = 0
    print('n:',n)

    while n > 0:
        print('--------')
        print('n - %:', n % 10 )
        n = n // 10
        print('n - //:', n)
        d = d + 1
        print('Iterations:', d)
    return d
```

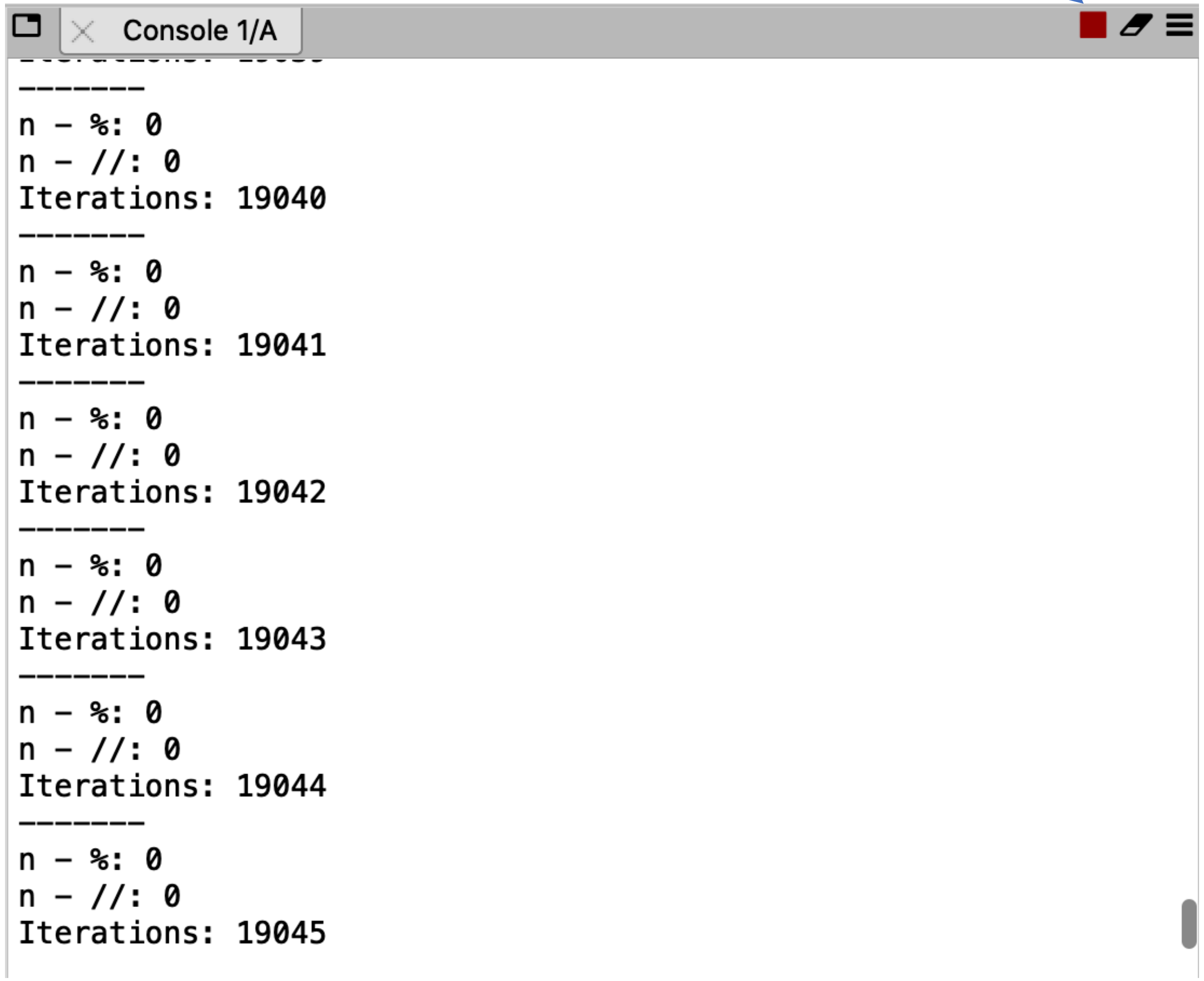
An improved version (deal with 0 and negative numbers)

Forever looping → Interrupt the code with Spyder

This would loop forever!

```
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    d = 0
    print('n:',n)

    while n >= 0:
        print('-------')
        print('n - %:', n % 10 )
        n = n // 10
        print('n - //:', n)
        d = d + 1
        print('Iterations:', d)
    return d
```

Click here to interrupt a running code!

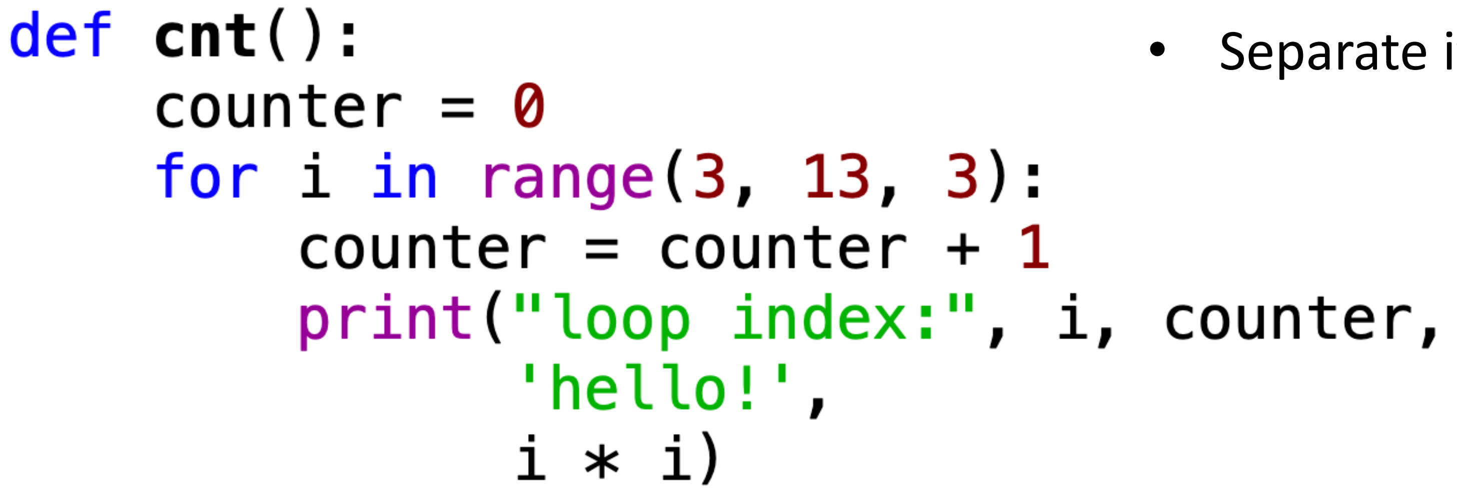


```
-------
n - %: 0
n - //: 0
Iterations: 19040
-------
n - %: 0
n - //: 0
Iterations: 19041
-------
n - %: 0
n - //: 0
Iterations: 19042
-------
n - %: 0
n - //: 0
Iterations: 19043
-------
n - %: 0
n - //: 0
Iterations: 19044
-------
n - %: 0
n - //: 0
Iterations: 19045
```

print() for online code debugging

```
counter = 0
for i in range(3, 13, 3):
   print('Loop index:', i)
   counter = counter + 1
print('Number of iterations:', counter, 'Final Loop index:', i )
```

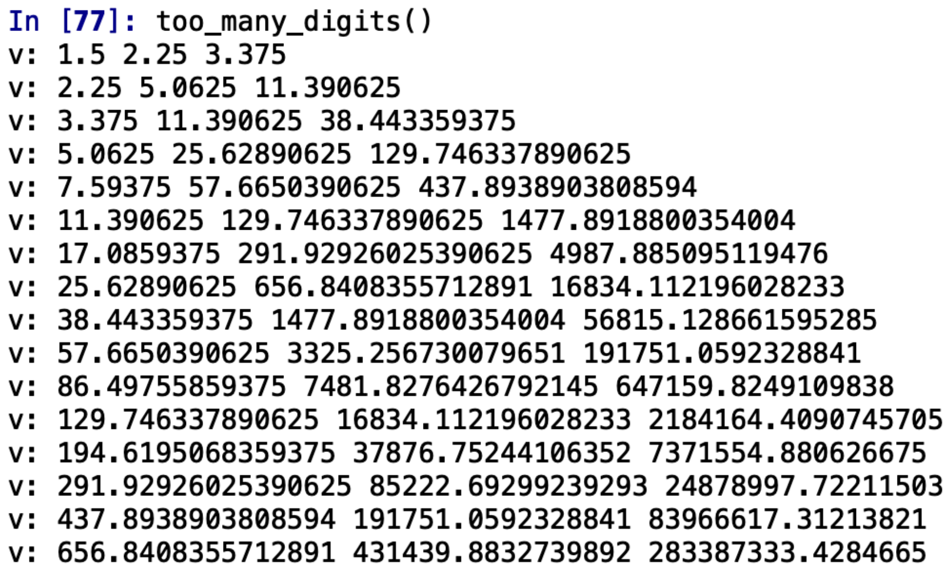
- Use print() to print anything useful to trace program behavior
- Separate items (string, variables) by commas



```
def cnt():
    counter = 0
    for i in range(3, 13, 3):
        counter = counter + 1
        print("loop index:", i, counter,
              'hello!',
              i * i)
```

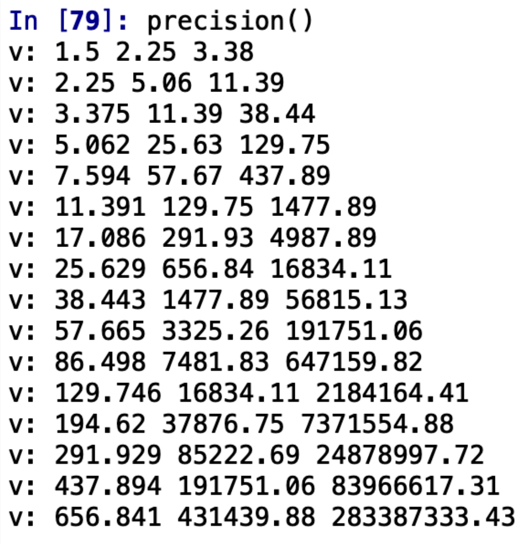
print() with round()

```
def too_many_digits():
    v = 1
    for i in range(3, 50, 3):
        v = v + ( v / 2)
        print("v:", v, v*v, v ** 3)
```



```
In [77]: too_many_digits()
v: 1.5 2.25 3.375
v: 2.25 5.0625 11.390625
v: 3.375 11.390625 38.443359375
v: 5.0625 25.62890625 129.746337890625
v: 7.59375 57.6650390625 437.8938903808594
v: 11.390625 129.746337890625 1477.8918800354004
v: 17.0859375 291.92926025390625 4987.885095119476
v: 25.62890625 656.8408355712891 16834.112196028233
v: 38.443359375 1477.8918800354004 56815.128661595285
v: 57.6650390625 3325.256730079651 191751.0592328841
v: 86.49755859375 7481.827642679214 5647159.8249109838
v: 129.746337890625 16834.112196028233 2184164.4090745705
v: 194.6195068359375 37876.75244106352 7371554.880626675
v: 291.92926025390625 85222.69299239293 24878997.72211503
v: 437.8938903808594 191751.0592328841 83966617.31213821
v: 656.8408355712891 431439.8832739892 283387333.4284665
```

```
def precision():
    v = 1
    for i in range(3, 50, 3):
        v = v + ( v / 2)
        print("v:", round(v,3),
              round(v*v, 2), round(v**3, 2))
```

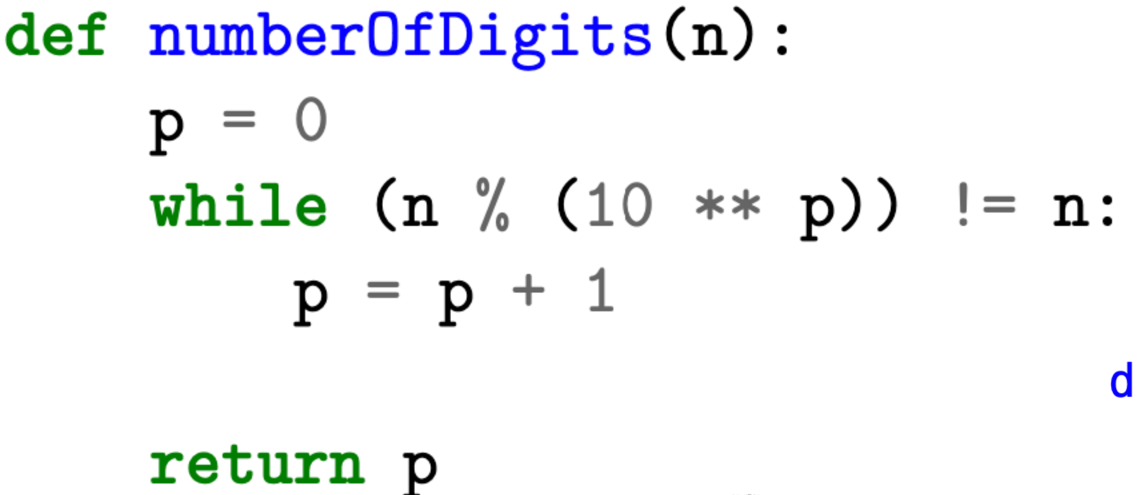


```
In [79]: precision()
v: 1.5 2.25 3.38
v: 2.25 5.06 11.39
v: 3.375 11.39 38.44
v: 5.062 25.63 129.75
v: 7.594 57.67 437.89
v: 11.391 129.75 1477.89
v: 17.086 291.93 4987.89
v: 25.629 656.84 16834.11
v: 38.443 1477.89 56815.13
v: 57.665 3325.26 191751.06
v: 86.498 7481.83 647159.82
v: 129.746 16834.11 2184164.41
v: 194.62 37876.75 7371554.88
v: 291.929 85222.69 24878997.72
v: 437.894 191751.06 83966617.31
v: 656.841 431439.88 283387333.43
```

What the code does?

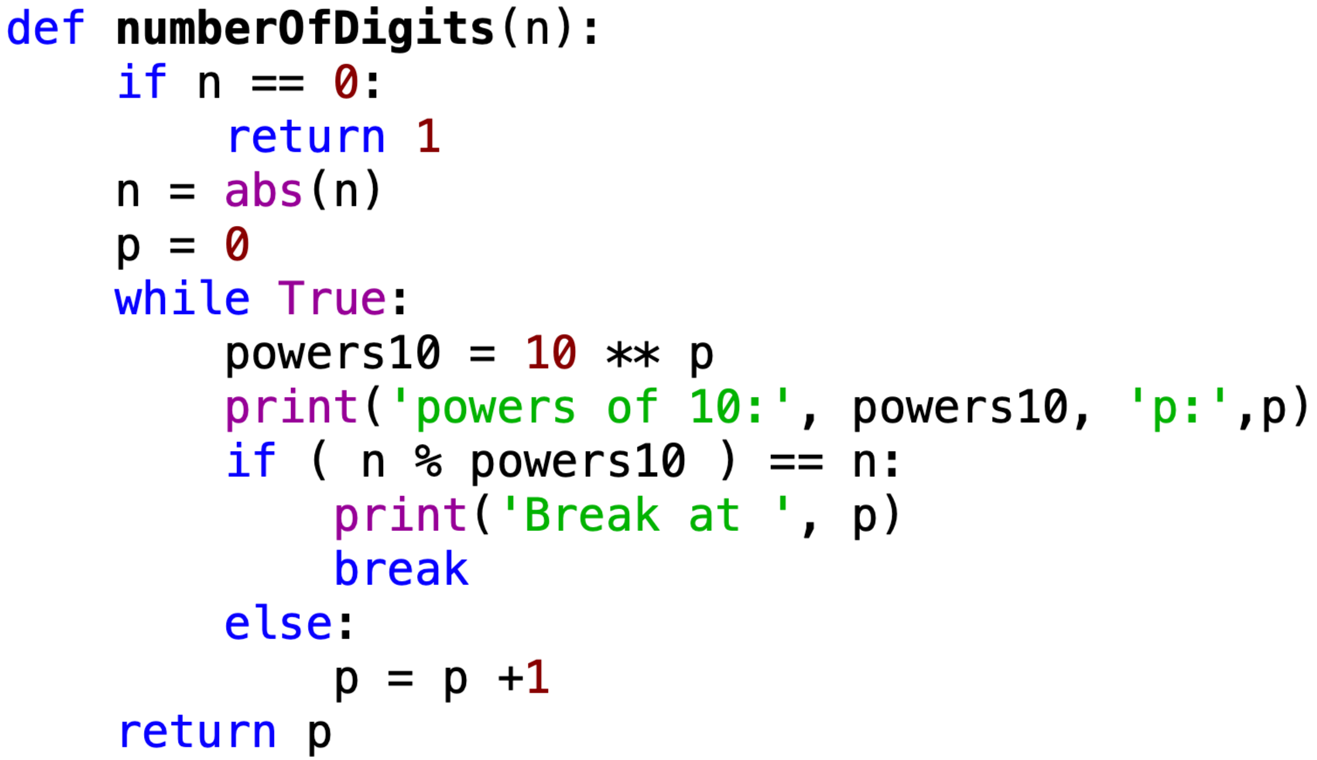
```
def [redacted](n):
    p = 0
    while (n % (10 ** p)) != n:
        p = p + 1

    return p
```

Two equivalent ways: while condition vs. while True + break

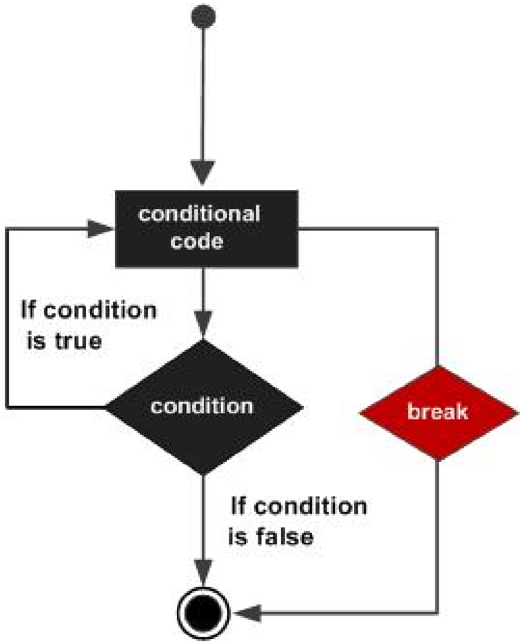


```
def numberOfDigits(n):
    p = 0
    while (n % (10 ** p)) != n:
        p = p + 1

    return p
```

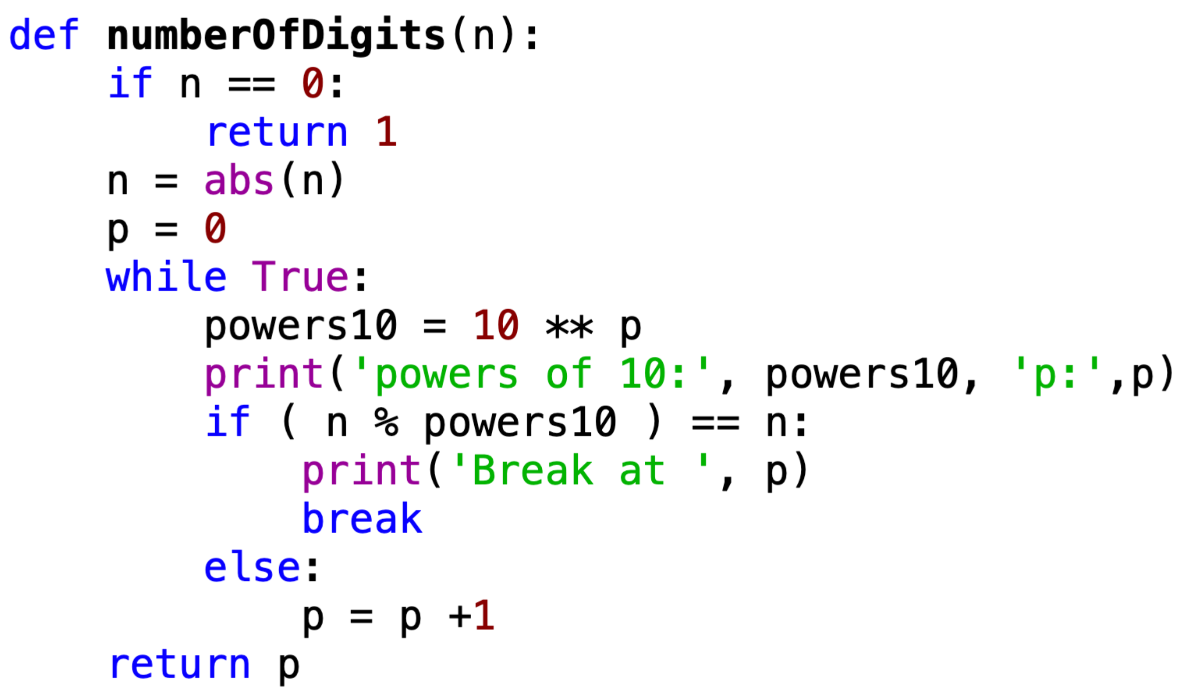
The same, but more decomposed



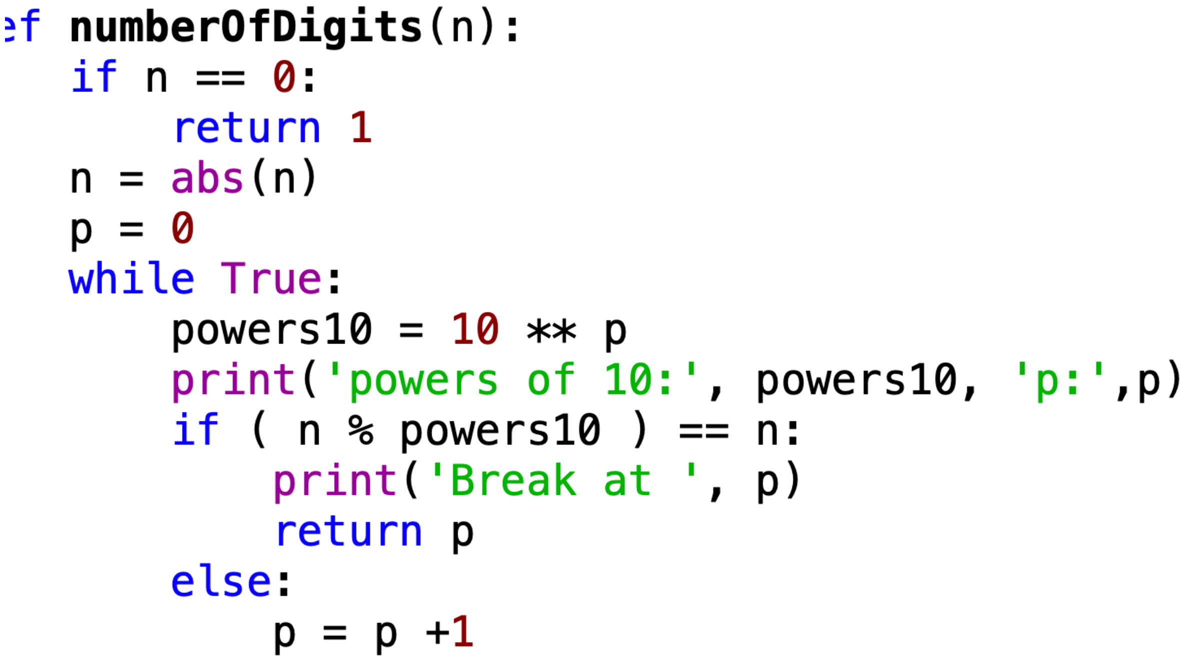
```
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    p = 0
    while True:
        powers10 = 10 ** p
        print('powers of 10:', powers10, 'p:',p)
        if ( n % powers10 ) == n:
            print('Break at ', p)
            break
        else:
            p = p +1
    return p
```



break vs. return



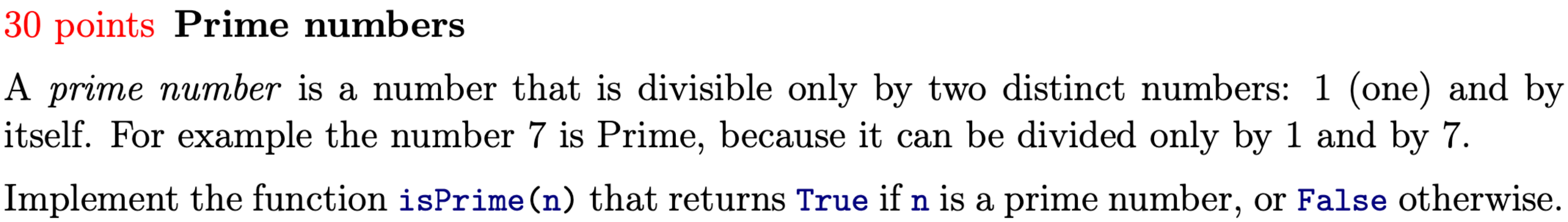
```
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    p = 0
    while True:
        powers10 = 10 ** p
        print('powers of 10:', powers10, 'p:',p)
        if ( n % powers10 ) == n:
            print('Break at ', p)
            break
        else:
            p = p +1
    return p
```



```
ef numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    p = 0
    while True:
        powers10 = 10 ** p
        print('powers of 10:', powers10, 'p:',p)
        if ( n % powers10 ) == n:
            print('Break at ', p)
            return p
        else:
            p = p +1
```

Do the same thing: break and return, or directly return

From Lab03



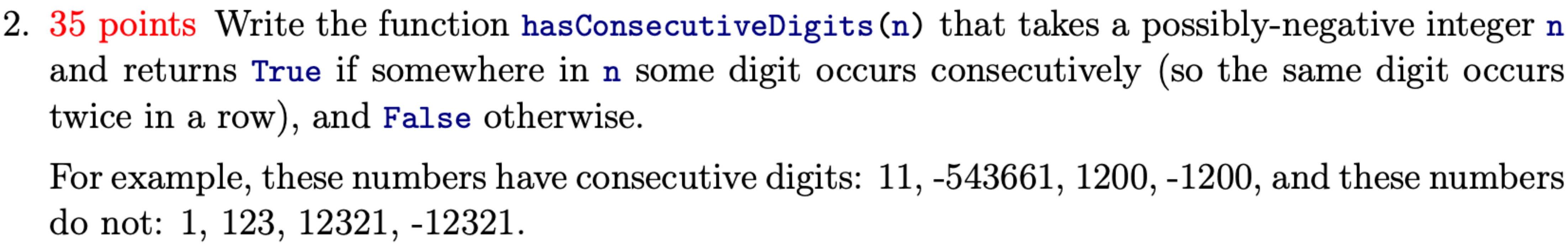
30 points **Prime numbers**

A *prime number* is a number that is divisible only by two distinct numbers: 1 (one) and by itself. For example the number 7 is Prime, because it can be divided only by 1 and by 7.

Implement the function `isPrime(n)` that returns `True` if `n` is a prime number, or `False` otherwise.

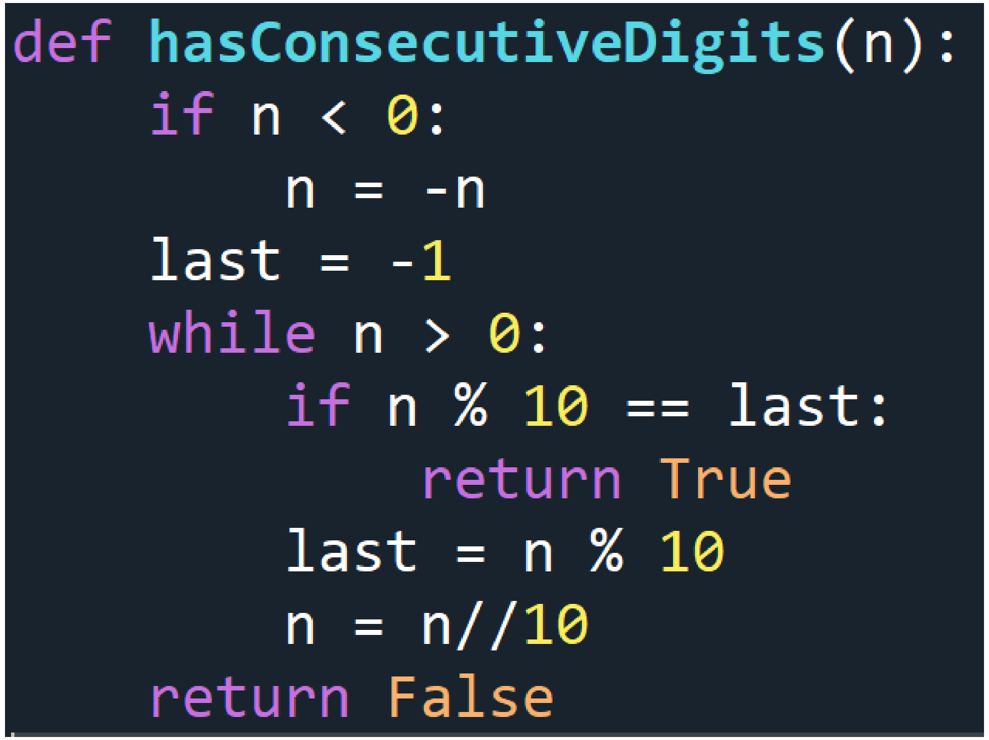
```
def isPrime(n):
    if n == 0 or n == 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

From Lab04



2. 35 points Write the function `hasConsecutiveDigits(n)` that takes a possibly-negative integer `n` and returns `True` if somewhere in `n` some digit occurs consecutively (so the same digit occurs twice in a row), and `False` otherwise.

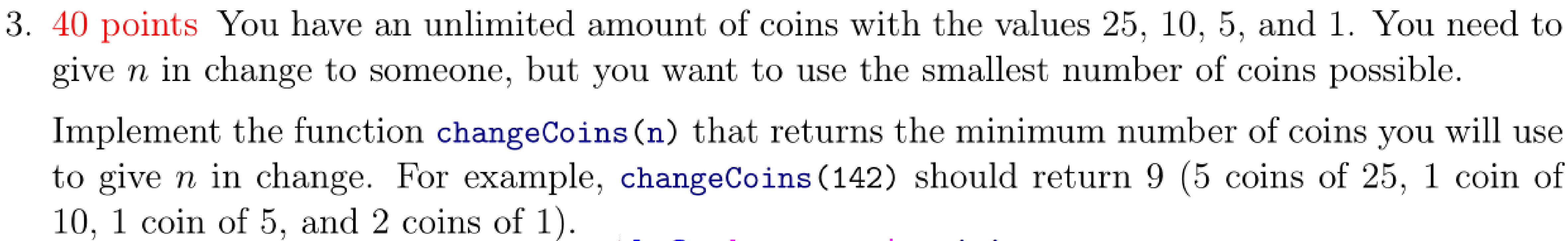
   For example, these numbers have consecutive digits: 11, -543661, 1200, -1200, and these numbers do not: 1, 123, 12321, -12321.



```python
def hasConsecutiveDigits(n):
    if n < 0:
        n = -n
    last = -1
    while n > 0:
        if n % 10 == last:
            return True
        last = n % 10
        n = n//10
    return False
```

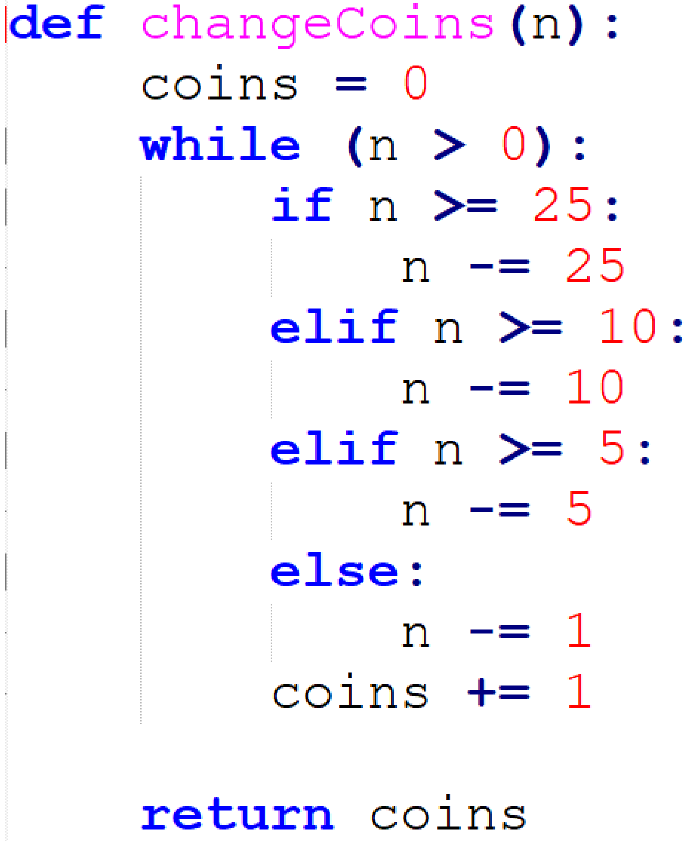
Use print() to understand what's going on!

From Lab04



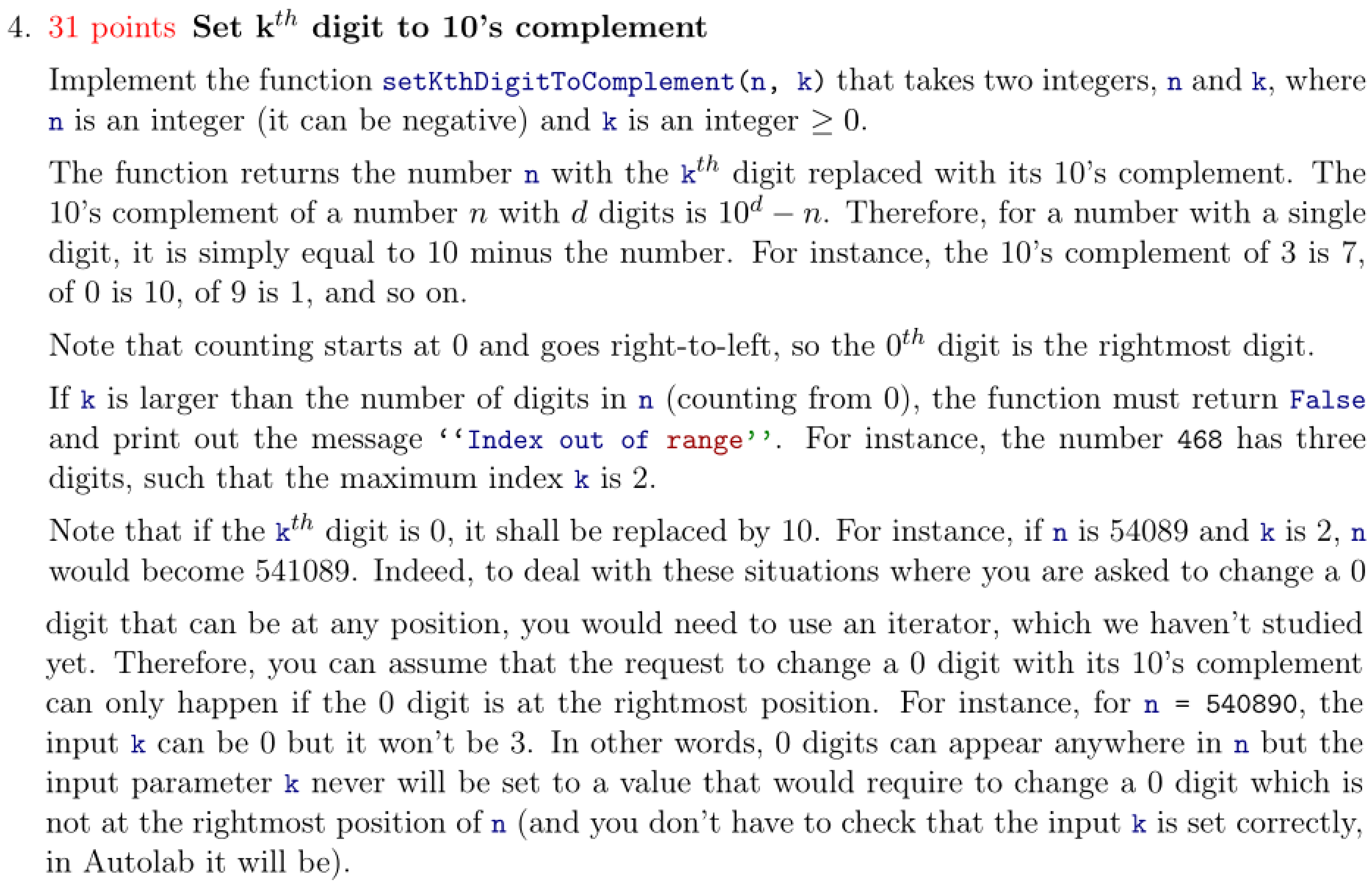
3. 40 points You have an unlimited amount of coins with the values 25, 10, 5, and 1. You need to give n in change to someone, but you want to use the smallest number of coins possible.

Implement the function `changeCoins(n)` that returns the minimum number of coins you will use to give n in change. For example, `changeCoins(142)` should return 9 (5 coins of 25, 1 coin of 10, 1 coin of 5, and 2 coins of 1).



```
def changeCoins(n):
    coins = 0
    while (n > 0):
        if n >= 25:
            n -= 25
        elif n >= 10:
            n -= 10
        elif n >= 5:
            n -= 5
        else:
            n -= 1
        coins += 1

    return coins
```

From HW03



4. 31 points **Set kth digit to 10’s complement**

Implement the function `setKthDigitToComplement(n, k)` that takes two integers, `n` and `k`, where `n` is an integer (it can be negative) and `k` is an integer ≥ 0.

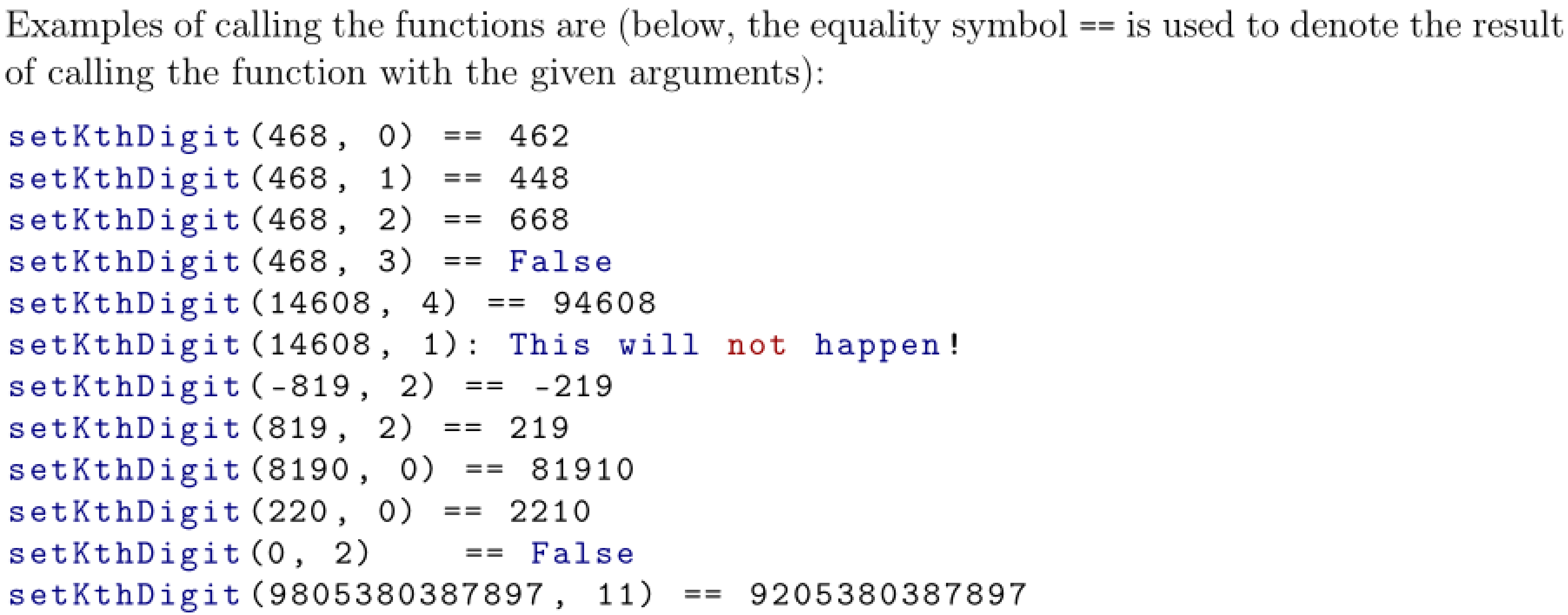
The function returns the number `n` with the `k`th digit replaced with its 10’s complement. The 10’s complement of a number n with d digits is $10^d - n$. Therefore, for a number with a single digit, it is simply equal to 10 minus the number. For instance, the 10’s complement of 3 is 7, of 0 is 10, of 9 is 1, and so on.

Note that counting starts at 0 and goes right-to-left, so the 0th digit is the rightmost digit.

If `k` is larger than the number of digits in `n` (counting from 0), the function must return `False` and print out the message ‘‘`Index out of range`’’. For instance, the number `468` has three digits, such that the maximum index `k` is 2.

Note that if the `k`th digit is 0, it shall be replaced by 10. For instance, if `n` is 54089 and `k` is 2, `n` would become 541089. Indeed, to deal with these situations where you are asked to change a 0 digit that can be at any position, you would need to use an iterator, which we haven’t studied yet. Therefore, you can assume that the request to change a 0 digit with its 10’s complement can only happen if the 0 digit is at the rightmost position. For instance, for `n = 540890`, the input `k` can be 0 but it won’t be 3. In other words, 0 digits can appear anywhere in `n` but the input parameter `k` never will be set to a value that would require to change a 0 digit which is not at the rightmost position of `n` (and you don’t have to check that the input `k` is set correctly, in Autolab it will be).

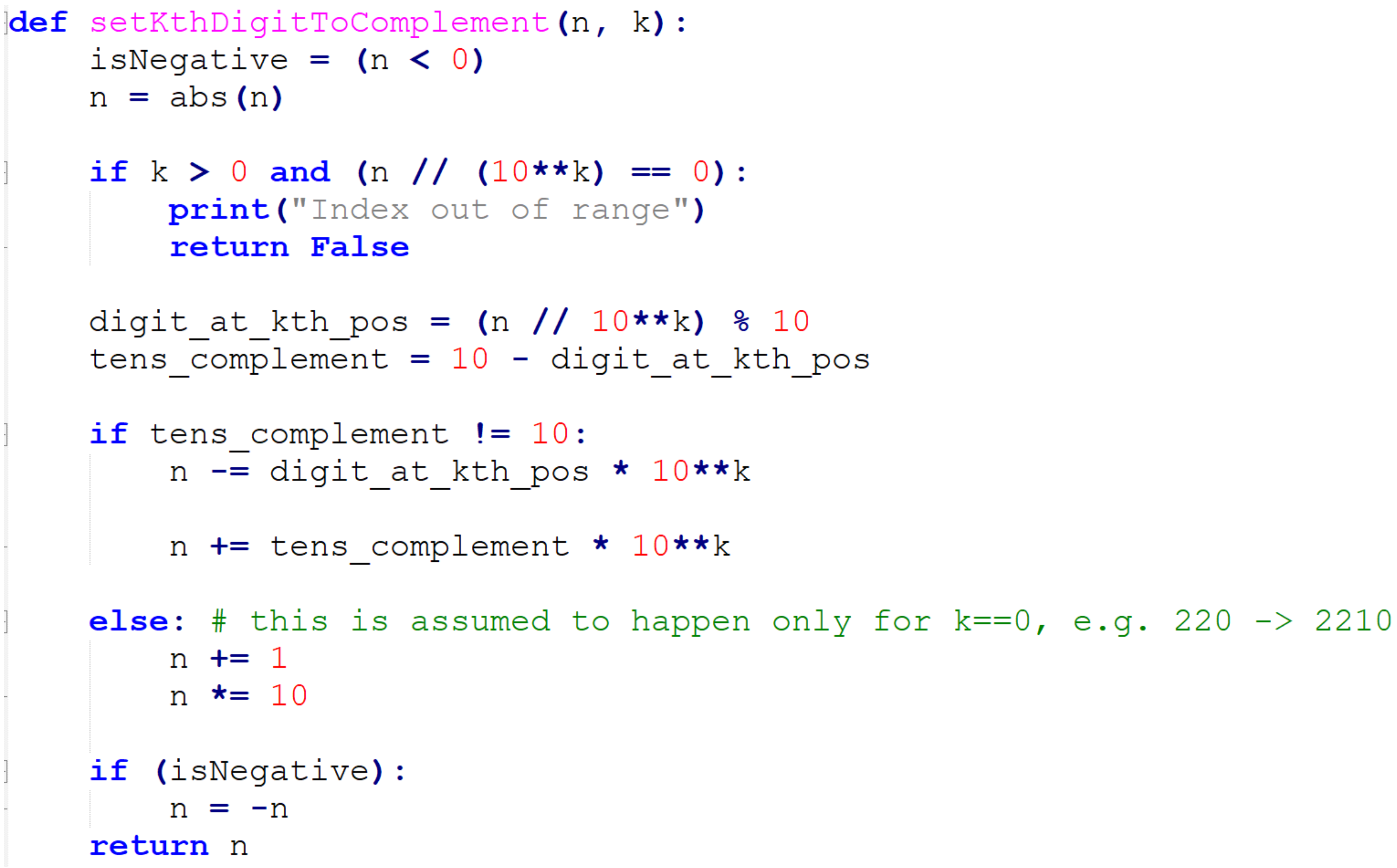
From HW03



Examples of calling the functions are (below, the equality symbol == is used to denote the result of calling the function with the given arguments):

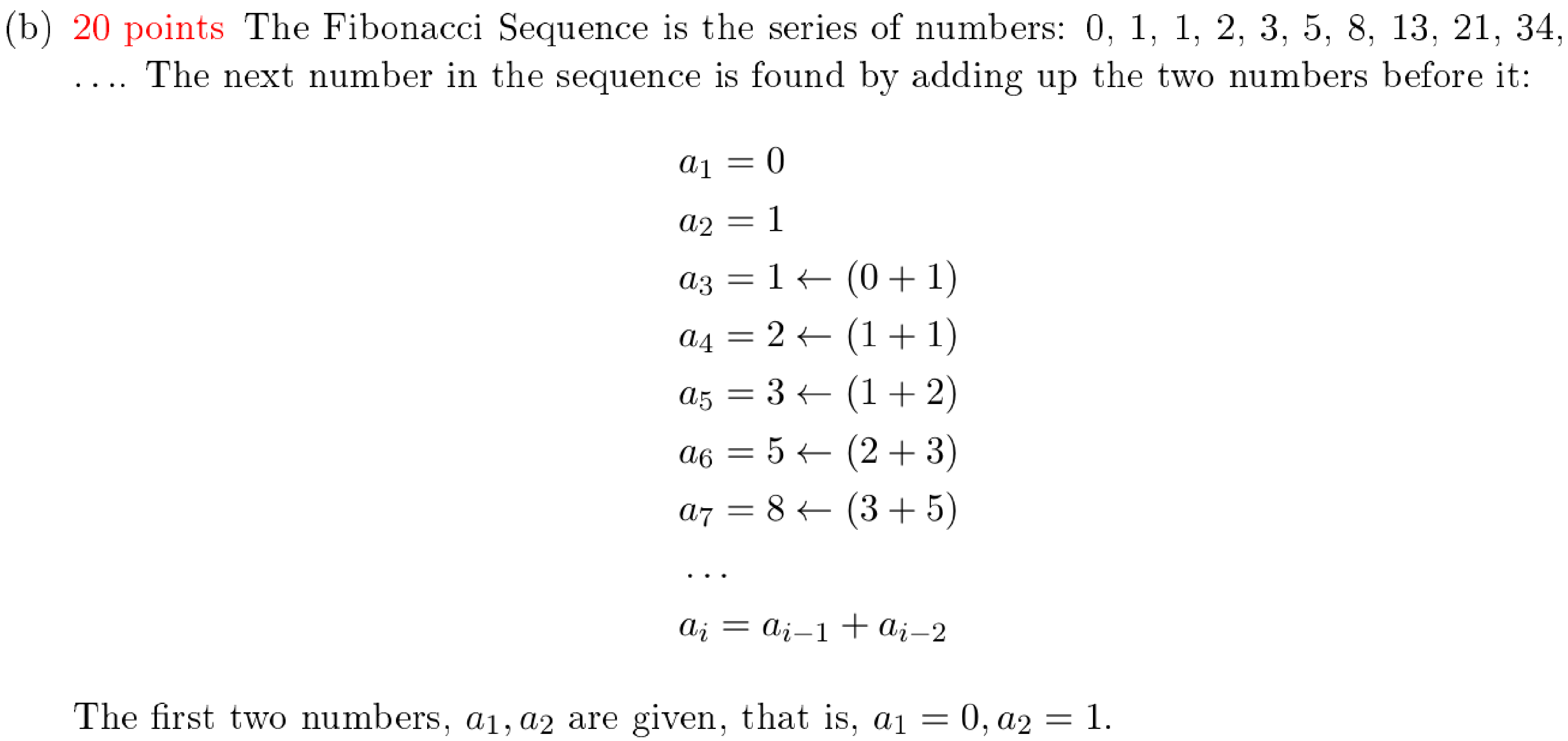
```
setKthDigit(468, 0) == 462
setKthDigit(468, 1) == 448
setKthDigit(468, 2) == 668
setKthDigit(468, 3) == False
setKthDigit(14608, 4) == 94608
setKthDigit(14608, 1): This will not happen!
setKthDigit(-819, 2) == -219
setKthDigit(819, 2) == 219
setKthDigit(8190, 0) == 81910
setKthDigit(220, 0) == 2210
setKthDigit(0, 2)    == False
setKthDigit(9805380387897, 11) == 9205380387897
```

From HW03



```python
def setKthDigitToComplement(n, k):
    isNegative = (n < 0)
    n = abs(n)

    if k > 0 and (n // (10**k) == 0):
        print("Index out of range")
        return False

    digit_at_kth_pos = (n // 10**k) % 10
    tens_complement = 10 - digit_at_kth_pos

    if tens_complement != 10:
        n -= digit_at_kth_pos * 10**k

        n += tens_complement * 10**k

    else: # this is assumed to happen only for k==0, e.g. 220 -> 2210
        n += 1
        n *= 10

    if (isNegative):
        n = -n
    return n
```

From Midterm

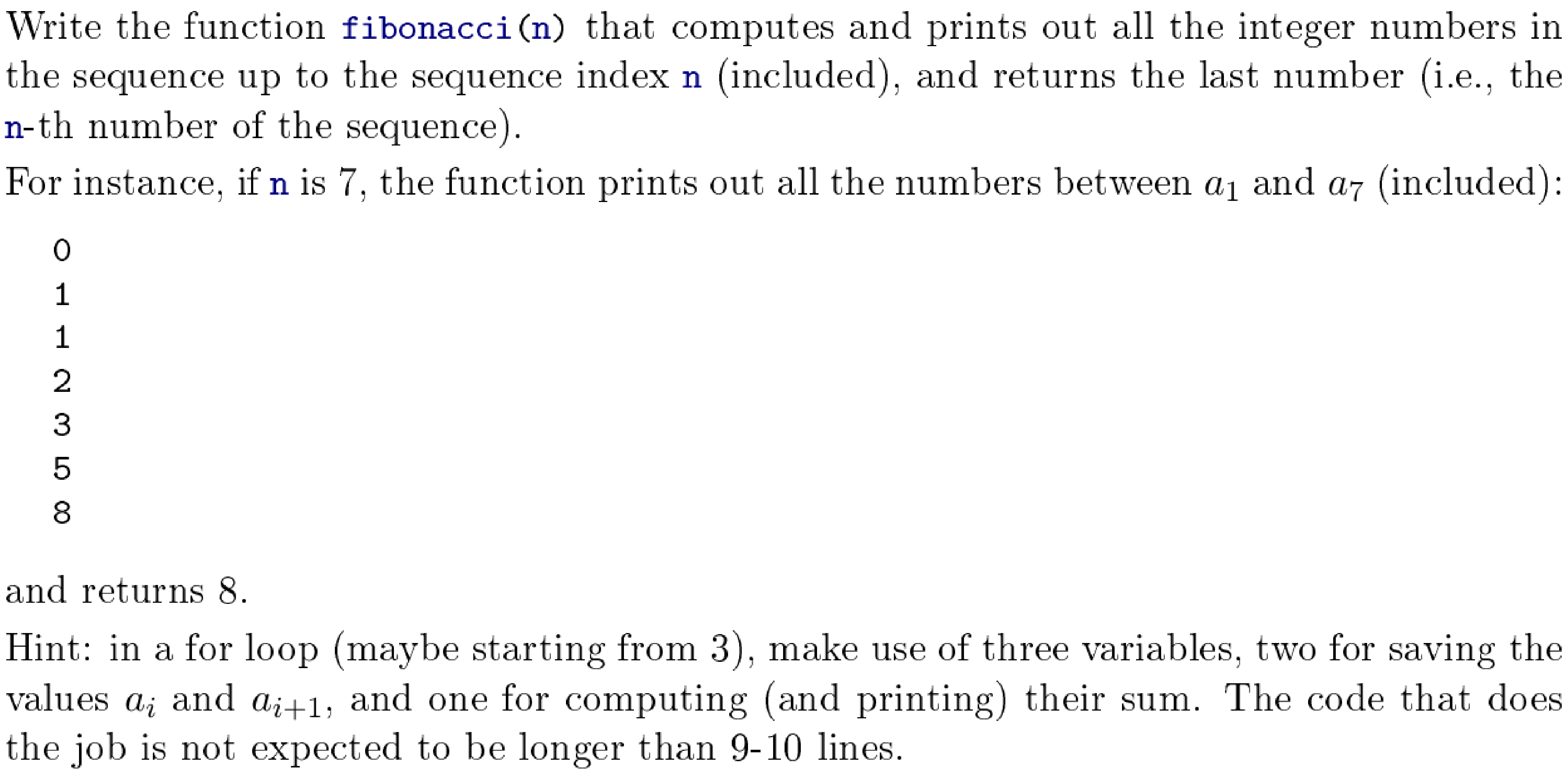


(b) 20 points The Fibonacci Sequence is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... The next number in the sequence is found by adding up the two numbers before it:

$$
\begin{aligned}
a_1 &= 0 \\
a_2 &= 1 \\
a_3 &= 1 \leftarrow (0+1) \\
a_4 &= 2 \leftarrow (1+1) \\
a_5 &= 3 \leftarrow (1+2) \\
a_6 &= 5 \leftarrow (2+3) \\
a_7 &= 8 \leftarrow (3+5) \\
&\dots \\
a_i &= a_{i-1} + a_{i-2}
\end{aligned}
$$

The first two numbers, a_1, a_2 are given, that is, $a_1 = 0, a_2 = 1$.

From Midterm



Write the function `fibonacci(n)` that computes and prints out all the integer numbers in the sequence up to the sequence index `n` (included), and returns the last number (i.e., the `n`-th number of the sequence).

For instance, if `n` is 7, the function prints out all the numbers between a_1 and a_7 (included):

```
0
1
1
2
3
5
8
```

and returns 8.

Hint: in a for loop (maybe starting from 3), make use of three variables, two for saving the values a_i and a_{i+1}, and one for computing (and printing) their sum. The code that does the job is not expected to be longer than 9-10 lines.