

fullName:_____ andrewID:_____ section:_____

15-112 F25
Midterm1 version B
80 Minutes

You **must write your name on this paper and hand this back** in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. **Do not unstaple any pages.** All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your exam. We will not grade anything on scrap paper.

You may not ask questions during the exam, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the exam, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use lists, sets, dictionaries, recursion, or anything we have not yet covered in class or the notes. Do not hardcode your answers. Assume `almostEqual(x, y)` and `rounded(n)` are both supplied for you. You must write all other helper functions you wish to use unless we specify otherwise.

Do not open this or look inside (even briefly) before the instructors signal for you to begin. When the instructors indicate that time is up, you must *immediately* stop writing, close this document, and hand it in. Do not look at anyone else's exam.

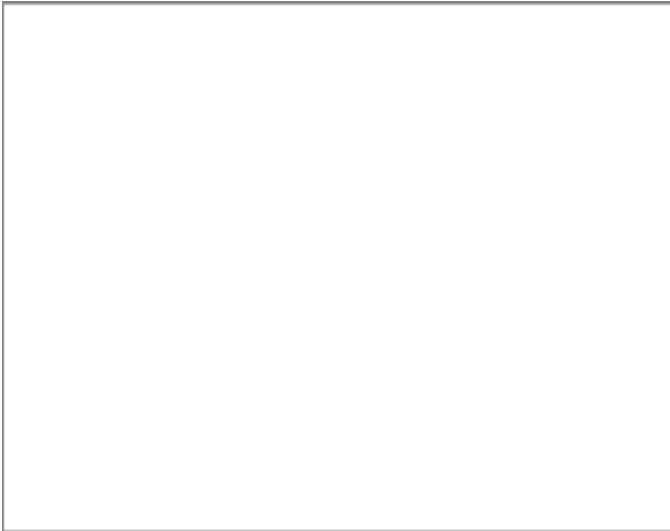
Code Tracing

CT1[10 pts]:

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
def ct1(x, y, z):  
    print(x * y % z * 2 + x ** z // y)  
    print(z / x)  
    y %= x  
    z += y  
    return 100 * x + 10 * y + z
```

```
print(ct1(2, 5, 3))
```



CT2[10 pts]:

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
import math

def ct2(x):
    y, z = math.floor(x), math.ceil(x)
    print('A', y, z)
    print('B', (z < bool(z)) or (y / 0 < 0))
    print('C', (y < z) and (z / 0 < 0))
    print('D', z / 0)

print(ct2(-3.2))
```



CT3[10 pts]:

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
def ct3(x):  
    r = x  
    while x < 10:  
        for y in range(x, x + 2):  
            x += y  
            r = 10 * r + y  
    print(r, x)  
  
print(ct3(2))
```



CT4[10 pts]:

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
def ct4(s):
    for i in range(len(s)):
        c = s[i]
        if c.isalpha():
            d = chr(ord(c) + i + 1)
            s = s.replace(c, d)
        else:
            s = s.replace(c, str(i))
    return s

print(ct4('AB!ab'))
```



Free Response

Your functions should work generally for the kinds of inputs specified in the problem statement, and we may test your code using additional test cases. We will manually grade free responses for partial credit if you do not pass all the test cases.

FR1[10pts]: kthDigitOfPi(k)

Background: `math.pi` equals 3.141592653589793, so it includes the first 16 digits of pi, where the 0th digit is 3 and the 1th digit is 1. With that in mind, write the one-line function `kthDigitOfPi(k)` that takes a non-negative int `k` and returns the `k`th digit of pi, or 'unsure' if `k > 15`. For partial credit, you may use more than one line (but still, no strings and no loops).

Notes:

1. You may not use strings or loops for this problem.
2. Do not use a semicolon (;) to place two statements on one line.
3. You may not hardcode the answer, so anything like the following is disallowed:

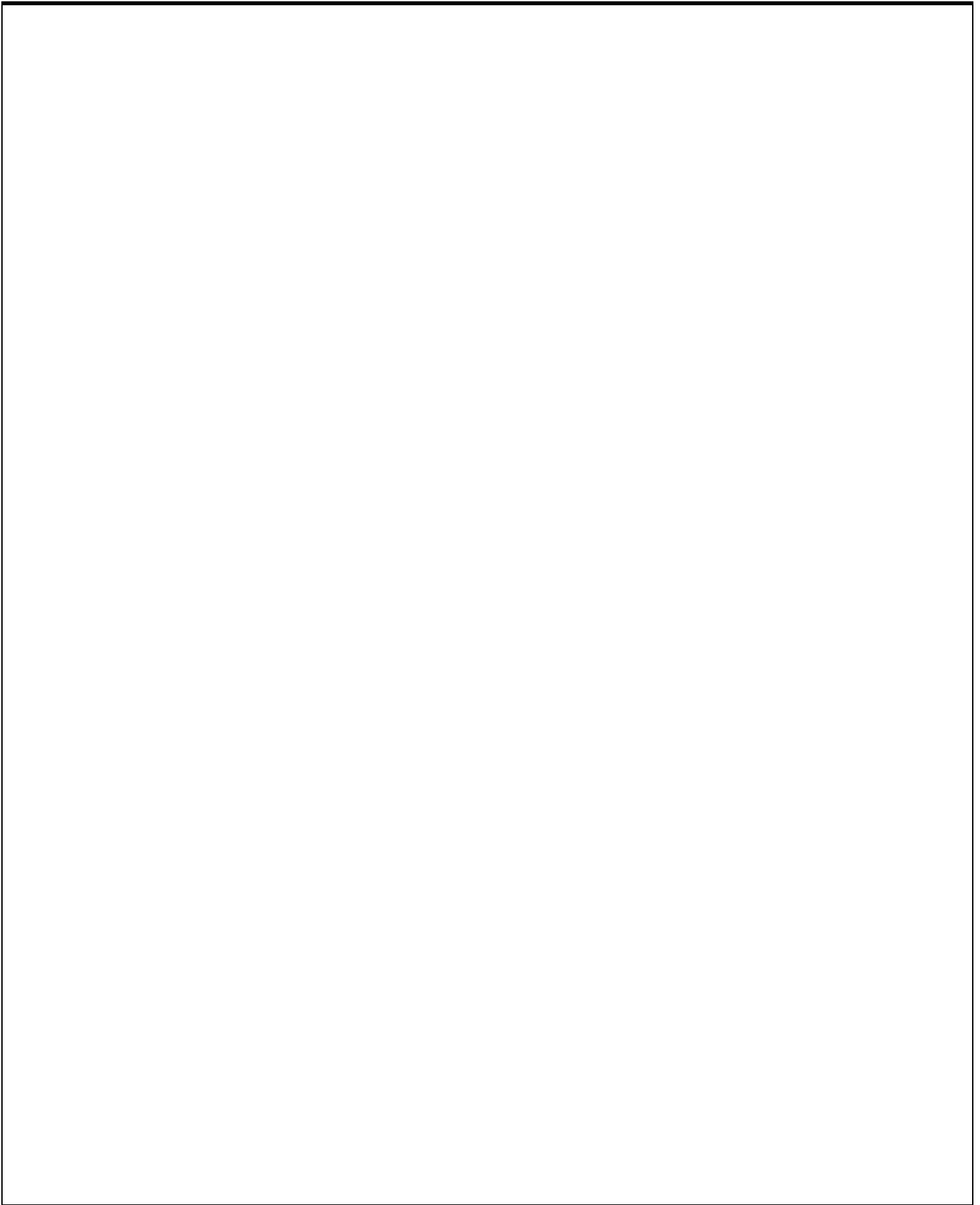
```
if k == 0: return 3
elif k == 1: return 1
...
```

Instead, you must perform some arithmetic on `math.pi` to find the answer.

Here are some test cases for you:

```
assert(kthDigitOfPi(0) == 3)
assert(kthDigitOfPi(1) == 1)
assert(kthDigitOfPi(2) == 4)
assert(kthDigitOfPi(3) == 1)
assert(kthDigitOfPi(4) == 5)
assert(kthDigitOfPi(14) == 9)
assert(kthDigitOfPi(15) == 3)
assert(kthDigitOfPi(16) == 'unsure')
```

Write your answer on the following page



FR2[25pts]: `fourify(n)` and `nthFourishPrime(n)`

Note: you may not use strings in either part of this problem.

Part 1: `fourify(n)`

Background: We will say that you can "fourify" (a coined term) an integer n by adding 4 to each digit in n , with wraparound at 10 as needed for each digit. For example, to fourify 508:

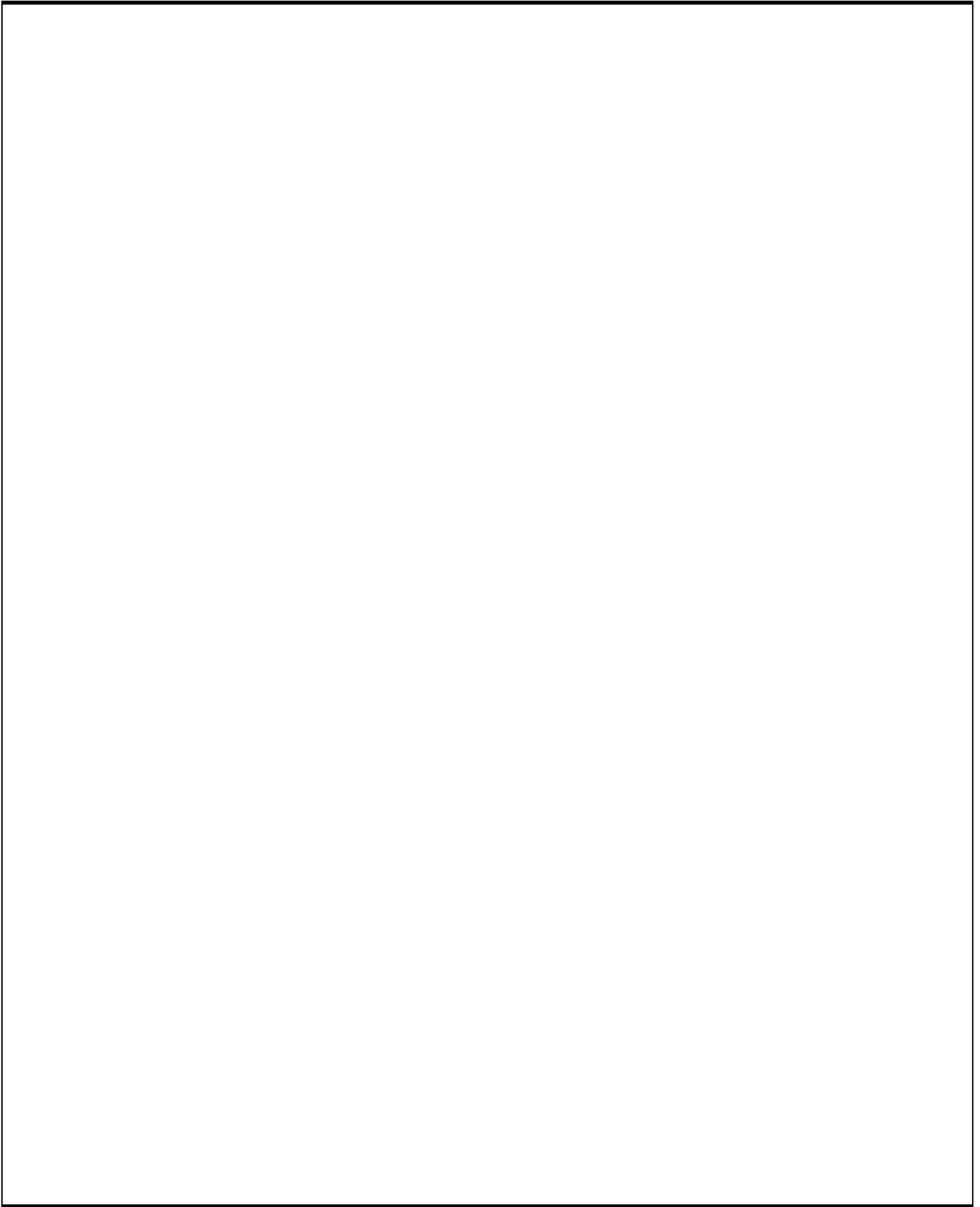
- * replace 5 with 9 (5+4)
- * replace 0 with 4 (0+4)
- * replace 8 with 2 (8+4 is 12, which we wraparound 10 to get 2)

Thus, we can fourify 508 to get 942.

Write the function `fourify(n)` that takes a positive integer n and returns the fourified version of n . For example:

```
assert(fourify(508) == 942)
assert(fourify(1234567890) == 5678901234)
```

Write your answer to part 1 (`fourify`) on the following page



FR2, Part 2: nthFourishPrime(n)

Note: this part uses fourify(n) that you wrote in Part 1, which you may assume works as intended here.

Here are the first several primes:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,...

We will say that a number n is a "fourish prime" (a coined term) if n is prime AND fourified n is also prime. For example, if $n = 23$, we see n is prime, and then we fourify n like so:

$$2 + 4 == 6$$

$$3 + 4 == 7$$

So we got 67, which is also prime, so 23 is a fourish prime.

For another example, if $n = 19$, we see n is prime, and then we fourify n like so:

$$1 + 4 == 5$$

$$9 + 4 == 13 \text{ which wraps around to } 3$$

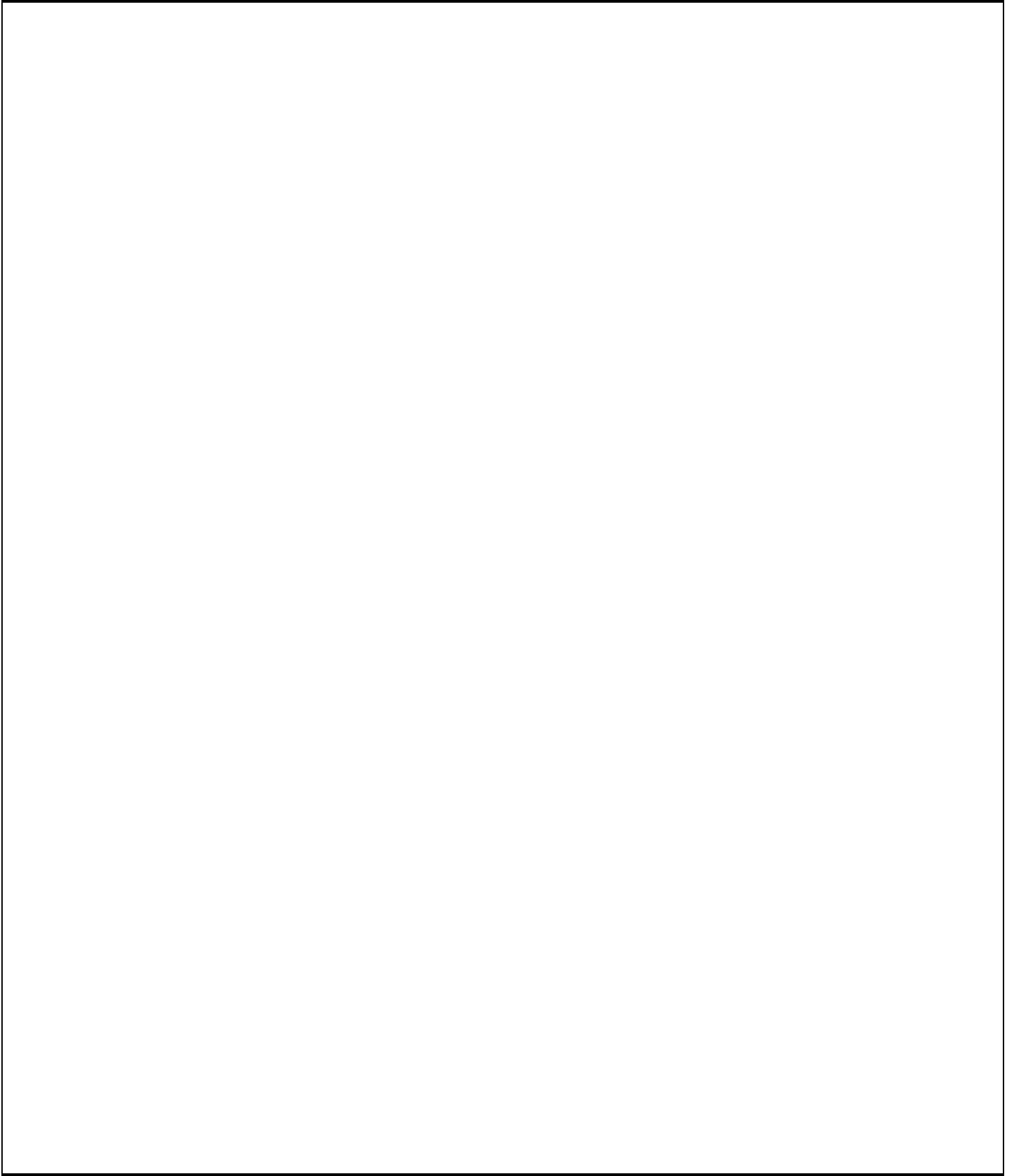
So we got 53, which is also prime, so 19 is a fourish prime.

With that in mind, write the function `nthFourishPrime(n)` that takes a non-negative int n and returns the n th fourish prime.

Here are some test cases for you:

```
assert(nthFourishPrime(0) == 3) # because 7 is also prime
assert(nthFourishPrime(1) == 19) # because 53 is also prime
assert(nthFourishPrime(2) == 23) # because 67 is also prime
assert(nthFourishPrime(3) == 37) # because 71 is also prime
assert(nthFourishPrime(4) == 53) # because 97 is also prime
assert(nthFourishPrime(5) == 61) # because 5 is also prime
```

Write your answer to part 2 (`nthFourishPrime`) on the following page



FR3[25pts]: encode(s) and decode(n)

Note: we found it very helpful to use strings even for digit manipulation, which is allowed here.

Part 1: encode(s)

Background: we will define an encoding of a string `s` as follows:

- * First, convert `s` to uppercase and ignore all the non-letters.
- * For each letter, convert the letter to its ascii value (where `ord('A')` is 65 and `ord('Z')` is 90, so every such code is between 65 and 90 inclusive).
- * At this point, you have a series of ascii codes each between 65 and 90.
- * Now, create two numbers. The first is formed by the tens digit of each ascii code. The second is formed by the ones digit of each ascii code.
- * Return a single int that consists of the digits of the first number followed by the digits of the second number.

For example, say we are encoding 'CT'. We start with this:

- * 'C' has ascii code 67
- * 'T' has ascii code 84

Now, here are the tens digits of those ascii codes:

- * 68

And here are the ones digits of those ascii codes:

- * 74

Finally, we concatenate those to get the result:

- * 6874

Thus, `encode('CT')` returns 6874.

Here is another example where `s` contains lowercase and non-letters.

Say we are encoding 'Cat?'. We start with this:

- * 'C' has ascii code 67
- * 'A' has ascii code 65
- * 'T' has ascii code 84
- * '?' is not a letter so we ignore it

Now, here are the tens digits of those ascii codes:

- * 668

And here are the ones digits of those ascii codes:

- * 754

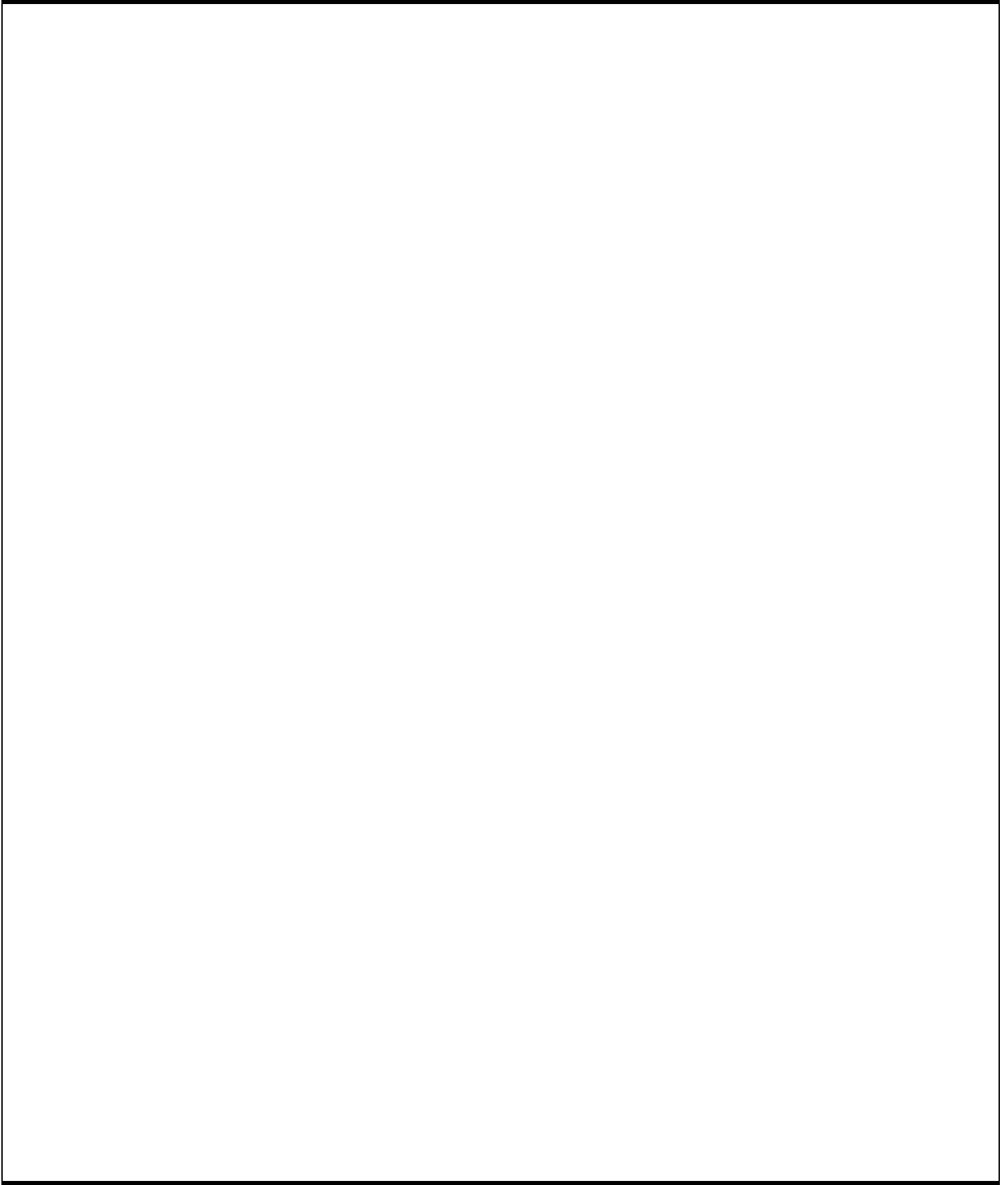
Finally, we concatenate those to get the result:

- * 668754

Thus, `encode('Cat?')` returns 668754.

With that in mind, write the function `encode(s)` that takes a string `s` and returns the integer encoding of `s` as just described. If there are no letters in `s`, return 0. If it helps, you may see some test cases below, after part 2.

Write your answer to part 1 (encode) on the following page



FR3, Part 2: decode(n)

Also write the function `decode(n)` that takes an integer encoding as described in Part 1, and returns the string `s` that is encoded.

Note that this string will be in uppercase and will only contain letters.
If `n` is 0, return the empty string.

Here are some test cases for you:

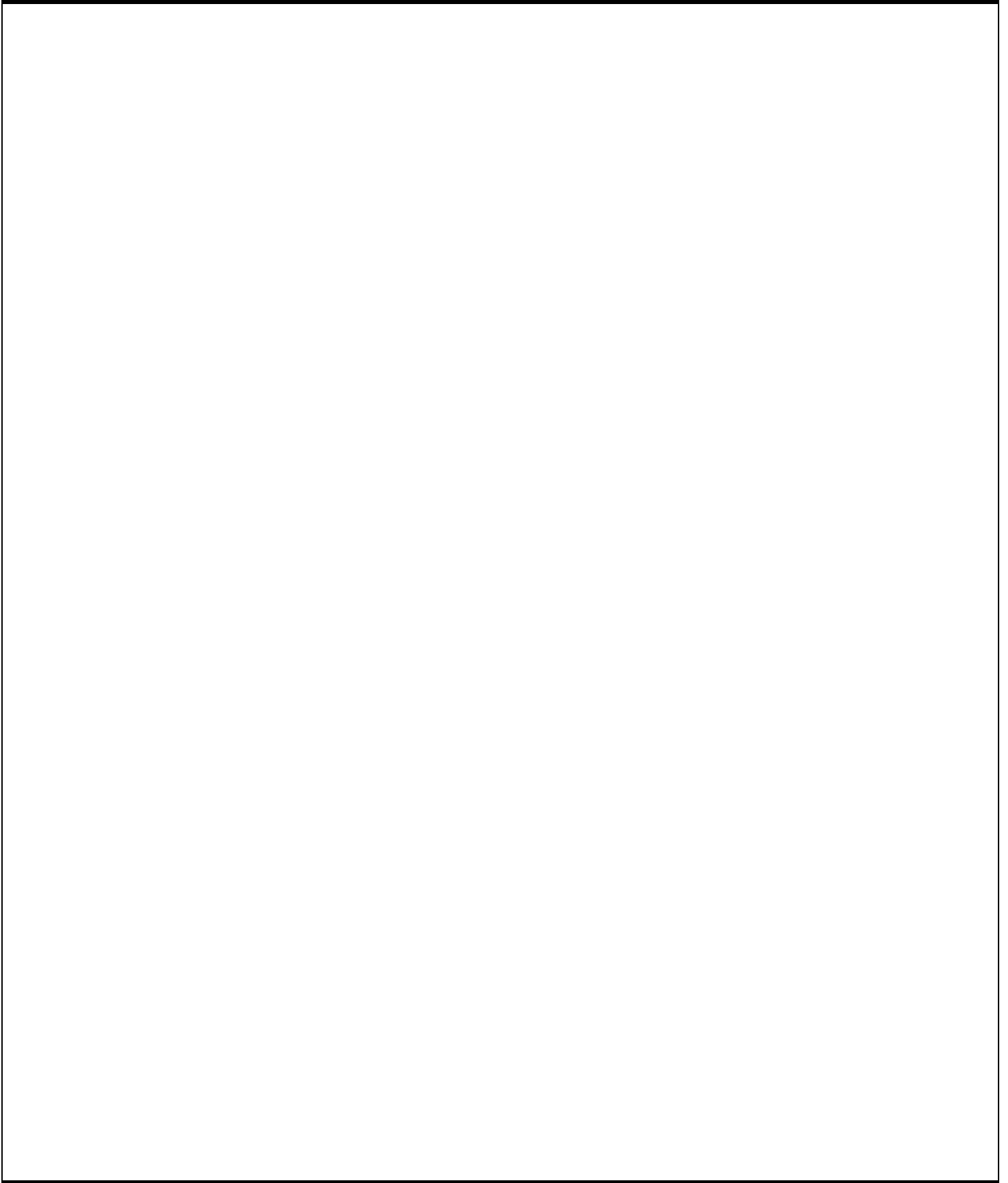
```
assert(encode('CT') == 6874) # C is 67, T is 84
assert(decode(6874) == 'CT')
```

```
assert(encode('Cat?') == 668754) # C is 67, A is 65, T is 84
assert(decode(668754) == 'CAT')
```

```
assert(encode('#C-d.eF#') == 66677890)
assert(decode(66677890) == 'CDEF') # C is 67, D is 68, E is 69, F is 70
```

```
assert(encode('') == 0)
assert(decode(0) == '')
```

Write your answer to part 2 (decode) on the following page



BonusCT

These CTs are optional, and intended to be very challenging. They are worth very few points. Indicate what the following code prints. Place your answers (and nothing else) in the boxes below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1[1pt]:

```
def bonusCt1(n, s):
    t = s[-1] + s
    for x in range(1, n):
        s *= x
    i = 1
    while t in s:
        i += 1
        s = s.replace(t, t[0])[:-1] + str(i)
    return s
print(bonusCt1(4, 'ef'))
```

bonusCT2[1pt]:

```
def bonusCt2(n):
    def f(n):
        return 1 if n < 10 else 1 + f(n//10)
    def g(n, k):
        return True if not n else n%10 == k and g(n//10, k)
    for c in range(123456789):
        if g(c, f(c)): n -= 1
        if not n: return c
print(bonusCt2(6))
```