Lab 1 Solutions lab01.zip (lab01.zip)

Solution Files

Required Questions

Review

What Would Python Display? (WWPD)

Q1: Return and Print

Use Ok to test your knowledge with the following "What Would Python Display?" questions:

```
python3 ok -q return-and-print -u
```

```
>>> def welcome():
...     print('Go')
...     return 'hello'
...
>>> def cal():
...     print('Bears')
...     return 'world'
...
>>> welcome()
------

>>> print(welcome(), cal())
------
```

Write Code

Q2: Debugging Quiz

The following is a quick quiz on different debugging techniques that will be helpful for you to use in this class. You can refer to the debugging article (../../articles/debugging/) to answer the questions.

Use Ok to test your understanding:

```
python3 ok -q debugging-quiz -u
```

Q3: Pick a Digit

Implement `digit`, which takes positive integers `n` and `k` and has only a single return statement as its body. It returns the digit of `n` that is `k` positions to the left of the rightmost digit (the one's digit). If `k` is 0, return the rightmost digit. If there is no digit of `n` that is `k` positions to the left of the rightmost digit, return 0.

Hint: Use `//` and `%` and the built-in `pow` function to isolate a particular digit of `n`.

```
def digit(n, k):
    """Return the digit that is k from the right of n for positive integers n and k.

    >>> digit(3579, 2)
    5
    >>> digit(3579, 0)
    9
    >>> digit(3579, 10)
    0
    """
    return n // pow(10, k) % 10
```

Use Ok to test your code:

```
python3 ok -q digit
```

Q4: Middle Number

Implement `middle` by writing a single return expression that evaluates to the value that is neither the largest or smallest among three different integers `a`, `b`, and `c`.

Hint: Try combining all the numbers and then taking away the ones you don't want to return.

```
def middle(a, b, c):
    """Return the number among a, b, and c that is not the smallest or largest.
    Assume a, b, and c are all different numbers.

    >>> middle(3, 5, 4)
    4
    >>> middle(30, 5, 4)
    5
    >>> middle(3, 5, 40)
    5
    >>> middle(3, 5, 40)
    5
    >>> middle(30, 5, 40)
    30
    """
    return a + b + c - min(a, b, c) - max(a, b, c)
```

Use Ok to test your code:

```
python3 ok -q middle
```

Syllabus Quiz

Q5: Syllabus Quiz

Please fill out the Syllabus Quiz (https://go.cs61a.org/syllabus-quiz), which confirms your understanding of the policies on the syllabus page (linked in the toolbar above).

Check Your Score Locally

You can locally check your score on each question of this assignment by running

```
python3 ok --score
```

This does NOT submit the assignment! When you are satisfied with your score, submit the assignment to Gradescope to receive credit for it.

Submit Assignment

If you are in a regular section of CS 61A, fill out this lab attendance and feedback form (https://forms.gle/dHxj8gttNWRY6Ptm9). (If you are in the mega section, you don't need to fill out the form.)

Then, submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** Lab 00 (../lab00/#submit-with-gradescope) has detailed instructions.

Optional Questions

> These questions are optional. If you don't complete them, you will still receive credit for this assignment. They are great practice, so do them anyway!

After you've watched the lecture videos on Control (lecture 3), come back and try these practice problems! You're welcome to ask questions about them in this lab, a future lab, or office hours.

Q6: Falling Factorial

Let's write a function `falling`, which is a "falling" factorial that takes two arguments, `n` and `k`, and returns the product of `k` consecutive numbers, starting from `n` and working downwards. When `k` is 0, the function should return 1.

```
def falling(n, k):
    """Compute the falling factorial of n to depth k.

    >>> falling(6, 3)  # 6 * 5 * 4
    120
    >>> falling(4, 3)  # 4 * 3 * 2
    24
    >>> falling(4, 1)  # 4
    4
    >>> falling(4, 0)
    1
    """
    total, stop = 1, n-k
    while n > stop:
        total, n = total*n, n-1
    return total
```

Use Ok to test your code:

```
python3 ok -q falling
```

Q7: Divisible By k

Write a function `divisible_by_k` that takes positive integers `n` and `k`. It prints all positive integers less than or equal to `n` that are divisible by `k` from smallest to largest. Then, it returns how many numbers were printed.

```
def divisible_by_k(n, k):
    """
    >>> a = divisible_by_k(10, 2)  # 2, 4, 6, 8, and 10 are divisible by 2
    2
    4
    6
    8
    10
    >>> a
    5
    >>> b = divisible_by_k(3, 1)  # 1, 2, and 3 are divisible by 1
    1
    2
    3
    >>> b
    3
    >>> c = divisible_by_k(6, 7)  # There are no integers up to 6 divisible by 7
    >>> c
    0
    """
    count = 0
    i = 1
    while i <= n:
        if i % k == 0:
            print(i)
            count += 1
        i += 1
    return count
```

Use Ok to test your code:

```
python3 ok -q divisible_by_k
```

Q8: Sum Digits

Write a function that takes in a nonnegative integer and sums its digits. (Using floor division and modulo might be helpful here!)

```
def sum_digits(y):
    """Sum all the digits of y.

    >>> sum_digits(10) # 1 + 0 = 1
    1
    >>> sum_digits(4224) # 4 + 2 + 2 + 4 = 12
    12
    >>> sum_digits(1234567890)
    45
    >>> a = sum_digits(123) # make sure that you are using return rather than print
    >>> a
    6
    """
    total = 0
    while y > 0:
        total, y = total + y % 10, y // 10
    return total
```

Use Ok to test your code:

```
python3 ok -q sum_digits
```

Q9: WWPD: What If?

Use Ok to test your knowledge with the following "What Would Python Display?" questions:

```
python3 ok -q if-statements -u
```

Hint: `print` (unlike `return`) does *not* cause the function to exit.

```
>>> def ab(c, d):
...     if c > 5:
...         print(c)
...     elif c > 7:
...         print(d)
...     print('foo')
>>> ab(10, 20)
______
```

```
>>> def bake(cake, make):
...     if cake == 0:
...         cake = cake + 1
...         print(cake)
...     if cake == 1:
...         print(make)
...     else:
...         return cake
...     return make
>>> bake(0, 29)
______

>>> bake(1, "mashed potatoes")
______
```

Q10: Double Eights

Write a function that takes in a number and determines if the digits contain two adjacent 8s.

```
def double_eights(n):
    """Return true if n has two eights in a row.
    >>> double_eights(8)
    False
    >>> double_eights(88)
    True
    >>> double_eights(2882)
    True
    >>> double_eights(880088)
    True
    >>> double_eights(12345)
    False
    >>> double_eights(80808080)
    False
    """
    prev_eight = False
    while n > 0:
        last_digit = n % 10
        if last_digit == 8 and prev_eight:
            return True
        elif last_digit == 8:
            prev_eight = True
        else:
            prev_eight = False
        n = n // 10
    return False

# Alternate solution
def double_eights_alt(n):
    while n:
        if n % 10 == 8 and n // 10 % 10 == 8:
            return True
        n //= 10
    return False
```

Use Ok to test your code:

```
python3 ok -q double_eights
```