Problem Set 2

---

a. last_digit

Prompt

Write a function called `last_digit` that takes in an integer `n` and returns the last digit in the integer. This function should also treat negative numbers the same as positive numbers. So the calls `last_digit(3)` and `last_digit(-3)` should both return `3`.

For example, `last_digit(1238476)` would return 6 since that is the last digit in `n`.

Starter code

```
def last_digit(n):
```

Example Input + Output

input	output	notes
last_digit(1238476)	6	
last_digit(123)	3	
last_digit(-123)	3	Negative special case

Example Solution

▼ Click here for the solution

```
def last_digit(n):
  return abs(n) % 10
```

Notes

▼ Click here for more information

- What's being assessed: integer manipulation
- What's challenging: remembering that you can pull off the last digit with % operator
- Source/Inspired by: trying to think of an ‘easy’ int manipulation problem without loops
- Written by: Paul Pham

Task Complexity

Branch	NO
Loop	NO
Nested Branch	NO

b. last_two

Prompt

Write a function called `last_two` that, given a string `s`, returns the string made of its last two characters. For example, given the string `“Howdy”`, this function would return `“dy”`.

If the given string has a length of two or fewer characters, return the String itself; an example of this case would be that this function would return `“a”` with the input String `“a”`.

Starter code

```
def last_two(s):
```

Example Input + Output

input	output	notes
last_two('howdy')	'dy'	
last_two('on')	'on'	
last_two('a')	'a'	special return case
last_two('')	''	special return case

Example Solution

▼ Click here for the solution

```
def last_two(string):
  return string[-2:]
```

Notes

▼ Click here for more information

- What's being assessed: conditions (if, else), string manipulation with negative indices
- What's challenging: slicing into strings with negative indices (although this problem could be solved with the slice string[len(string) - 2:]
- Source/Inspired by: CodingBat problem first_two
- Written by: Paul Pham

Task Complexity

Branch	NO
Loop	NO
Nested Branch	NO

c. has_word

Prompt

Write a function called `has_word that` takes in two strings as parameters. Return the integer `True` if the first string parameter contains any instance of the second string parameter and `False` otherwise.

For example, the call `has_word('This is a sentence', 'sentence')` would return `True` since `'sentence'` is a substring of `'This is a sentence'`.

An example that returns `False` is the call `has_word('asdfghjkl', 'hello')` because `'hello'` is not in the string `'asdfghjkl'`.

Starter code

```
def has_word(long, short):
```

Example Input + Output

input	output	notes
has_word('this is a sentence', 'sentence')	1	
has_word('asdfghjkl', 'hello')	0	
has_word('Thank you!', 'k you')	1	

Example Solution

▼ Click here for the solution

```
def has_word(long, short):
  if short in long:
    return 1
  else:
    return 0
```

Notes

▼ Click here for more information

- What's being assessed: use of the `in` keyword
- What's challenging: remembering that you can use `in` as a `contains` for string
- Source/Inspired by: trying to think of an ‘easy’ string problem
- Written by: Paul Pham

Task Complexity

Branch	NO
Loop	NO
Nested Branch	NO

---

d. verify_employee

Prompt

Write a function called `verify_employee(role_one, role_two, salary_one, salary_two)` where all parameters are numbers and the function has the following rules:

1. return -1 if `role_one` is less than `role_two` and `salary_one` is also less than `salary_two`.
2. return 0 if `role_one` and `role_two` are the same and `salary_one` and `salary_two` are the same

3. return 1 if `role_one` is greater than `role_two` and `salary_one` is also greater than `salary_two`.
4. return 2 if none of the previous rules could be verified

Starter code

```
def verify_employee(role_one, role_two, salary_one, salary_two):
```

Example Input + Output

input	output	notes
verify_employee(1, 1, 0, 0)	0	
verify_employee(1, 2, 10, 100)	-1	
verify_employee(2, 1, 100, 90)	1	
verify_employee(2, 1, 90, 100)	2	
verify_employee(5, 4, 100, 100)	2	
verify_employee(1, 1, 100, 90)	2	
verify_employee(-2, -1, 10, 100)	-1	

Example Solution

▼ Click here for the solution

```
def verify_employee(role_one, role_two, salary_one, salary_two):
    if role_one < role_two and salary_one < salary_two:
        return -1
    elif role_one == role_two and salary_one == salary_two:
        return 0
    elif role_one > role_two and salary_one > salary_two:
        return 1
    else:
        return 2
```

Notes

▼ Click here for more information

- What's being assessed: conditions (if, elif, else), operators (<, >, and, ==)
- What's challenging: translating plain english to conditional operators
- Source/Inspired by: Item from Matt's NCME 2019 study
- Written by: Benji
- Don't need logical "or" for this item
- From Matt's study

Task Complexity

Branch	YES
Loop	NO
Nested Branch	YES

---

e. contains_max_diff

Prompt

Write a function called `contains_max_diff` that takes in a list of integers and returns `1` if the list contains the number that represents the difference between the largest and smallest numbers in the list and returns `0` otherwise.

For example, the call `contains_max_diff([10, 9, 5, 3, 4, 7])` would return `1` since the difference between the largest number (10) and the smallest number (3) is 7, which is in the given list.

If the length of the list is less than two, return `False`.

Starter code

```
def contains_max_diff(nums):
```

Example Input + Output

input	output	notes
contains_max_diff([10, 9, 5, 3, 4, 7])	True	
contains_max_diff([10, 9, 5, 3, 4]))	False	
contains_max_diff([10])	False	

Example Solution

▼ Click here for the solution

```
def contains_max_diff(nums):
    if len(nums) < 2:
        return False

    small = min(nums)
    big = max(nums)

    difference = big - small
    return difference in nums
```

Notes

▼ Click here for more information

- What's being assessed: use of built-in functions, arithmetic logic
- What's challenging: calculating difference,
- Source/Inspired by: CodingBat problem max_end3
- Written by: Paul Pham

Task Complexity

Branch	YES
Loop	NO
Nested Branch	NO

f. enough_candy

Prompt

Everyone loves candy, but you've only got so much to hand out for Halloween! Kids and teens come to your door asking for candy, so of course, you'll give it to them, but teenagers need more candy to get through high school.

Write a function called `enough_candy` that will return `True` if you had enough candy to last a night full of trick-or-treaters (kids and teens who come to your door asking for candy), or `False` otherwise.

Parameters:

- `total_candy` is an integer that represents the number of pieces of candy that you have
- `treaters` is a list of integers that contains only 0s and 1s that where each value represents someone who comes to your door; a `1` represents teens and a `0` represents kids.

This function has the following rules:

- Teens ask for 3 pieces of candy
- Kids ask for 1 piece of candy
- You have enough candy if you have 0 or more pieces of candy after helping everyone

Starter code

```
def enough_candy(total_candy, treaters):
```

Example Input + Output

input	output	notes
enough_candy(10, [1, 1, 0, 1])	1	
enough_candy(3, [1, 1])	0	
enough_candy(6, [1, 0, 0, 0])	1	

Example Solution

▼ Click here for the solution

```
def enough_candy(total_candy, treaters):
    for treater in treaters:
        if treater == 1:
            total_candy -= 3
        else:
            total_candy -= 1
    return total_candy >= 0
```

Notes

▼ Click here for more information

- What's being assessed: conditionals, operators, translating story problem into code
- What's challenging: multiple branches, traversing the data structure correctly
- Source/Inspired by: `can_buy_car`
- Written by: Paul Pham

Task Complexity

Branch	YES
Loop	YES
Nested Branch	YES

g. check_passcode

Prompt

Write a function `check_passcode()` that determines if the parameter `pin`, which is a 4-digit integer, is a valid passcode. `pin` is valid if the sum of the first 3 digits modulus 7 is equal to the last digit. If `pin` is valid, the function should return `True`. If the string is not valid, it should return `False`.

So if `pin` were set to 5312, it would be a valid passcode and your code would return `True` because the first 3 digits (5, 3, and 1) sum to 9, and 9 modulus 7 equals the last digit (2). 1234 would not be a valid passcode so the function would return `False`.

You may assume that the `pin` is always a 4-digit integer value.

Starter code

```
def check_passcode(pin):
```

Example Input + Output

input	output	notes
check_passcode(5312)	True	
check_passcode(1234)	False	
check_passcode(1236)	True	
check_passcode(9873)	True	
check_passcode(9876)	False	

Example Solution

▼ Click here for the solution

```
def password_check(pin):
  target = pin % 10
  pin //= 10
```

```
total = 0
while pin > 0:
  digit = pin % 10
  total += digit
  pin //= 10

return total % 7 == target
```

Notes

▼ Click here for more information

- What's being assessed: variable storage, conditional statement, mod operator
- What's challenging: processing each digit, modulo operator
- Source/Inspired by: CSE 2019 study post-test
- Written by: Benji Xie

▼ Rubric (out of 9 pts)

- 2 pts for storing digits (-0.5 for any incorrectly stored digit)
- 2 pt for using mod to access each digit
- 2 pt for conditional statement
- 1 pt for correct print
- 2 pt for syntax (-1 for minor mistakes, -2 for major)

Task Complexity

Branch	YES
Loop	YES
Nested Loop	NO