# Information Processing 1 — Week 7 exercises

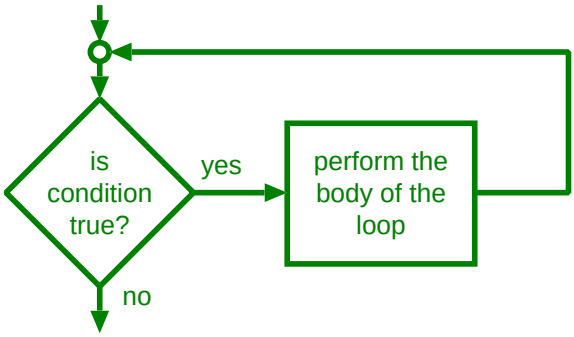
Complete each exercise on your own laptop before the end of the exercise class. When you finish an exercise, **immediately** ask an instructor to check your answer. Answers checked after the end of the class will have a 50% penalty.

**Do not** struggle for more than a few minutes with anything you cannot solve. If you are stuck or have any questions then **raise your hand** and an instructor will help you.

## 1 While loops and conditionals [4 points]

The `while` loop repeats one or more statement(s) as long as a condition is true.

```
while condition :
    statements to run
    repeatedly while
    condition is true
```




For example,

```
n = 1
while n < 1000:
    print(n)
    n = n * 2
print("done")
```

produces: `1 2 4 8 16 32 64 128 256 512`

### 1.1 Printing integers up to a given value

Use a `while` loop to implement a function `print_each(n)` that prints the integers from 0 to $n-1$. Verify that `print_each(10)` prints the digits from 0 to 9.

Hint: start with the example shown above and *simplify* it to make the solution to this question.

Make a copy of your function called `print_each1(x)`.

### 1.2 Accessing elements of a sequence by index

The built-in function `len(x)` tells you the length of the sequence `x` (which might be a string, for example).

Modify your function `print_each1(x)` so that it prints each of the elements in `x`, Print one element per line. Verify that `print_each1("hello")` prints the five letters in that word.

Make a copy of your function called `print_each2(x, n)`.

### 1.3 Accessing elements of a sub-sequence by index

Modify your function `print_each2(x, n)` to print the elements in `x` starting from a specified position `n`. (The first element in `x` is numbered 0.) Verify that `print_each2("goodbye", 4)` prints the last three letters of that word.

Make a copy of your function called `print_each3(x, n)`.

### 1.4 Default parameter values for common cases

Modify your function `print_each3(x, n)` so that the parameter `n` is optional. If `print_each3()` is called with only one argument, `x`, then the entire contents of `x` should be printed. Verify that `print_each3("goodbye", 4)` prints "bye" and that `print_each3("goodbye")` prints the entire word.

▷▷ Show your four functions (from `print_each` to `print_each3`) to an instructor **now**.

## 2 Identifying elements in an sequence [4 points]

The 'slice' operator '`[]`' can be used to obtain elements from a sequence. The simplest use is to obtain a single element from a sequence, treating the sequence as a array. For example:

```
x = "abcdef"

x[0]   # 'a'   non-negative indexes count from the first element towards the right
x[1]   # 'b'
x[2]   # 'c'
x[-1]  # 'f'   negative indexes count from the last element towards the left
x[-2]  # 'e'
x[-3]  # 'd'
```

### 2.1 Searching for an element

Use a `while` loop to write a function called `find_letter(word, letter)` that searches `word` (a string) for `letter` (a string containing only a single character). Your function should return the index of the first occurrence of `letter` in `word`. If `letter` does not occur in `word` then it should return `None`.

Verify that:

```
word = "abcdef"

print(find_letter(word, 'a'))    prints '0'
print(find_letter(word, 'f'))    prints '5'
print(find_letter(word, 'z'))    prints 'None'
```

Make a copy of your function called `find_letter1()`.

### 2.2 For loops

A `for` loop repeats a fixed number of times, each time setting a variable to the next value in a given sequence. For example

```
for i in range(10):
    print(i)
```

prints: 0 1 2 3 4 5 6 7 8 9 (The value of the expression '`range(10)`' is a sequence of 10 integers starting with 0.)

Rewrite your `find_letter1(word, letter)` function to use a `for` loop instead of a `while` loop.

Verify that it produces the same answers as above.

Make a copy of your function called `count_letter(word, letter)`.

### 2.3 Counting the occurrences of a character in a string

Write a function `count_letter(word, letter)` that counts how many times `letter` occurs in `word`. Verify that:

```
word = "bananas"
print(count_letter(word, 'a'))    prints '3'
print(count_letter(word, 'n'))    prints '2'
print(count_letter(word, 'z'))    prints '0'
```

Make a copy of your function called `count_letters(word, letters)`.

### 2.4 Nested loops

A `for` loop sets a variable to each value in a sequence. A string is a sequence of characters. A `for` loop can therefore set a variable to each character in a string. For example,

```
for c in "hello":
    print(c)
```

prints: `h e l l o`

Generalise the behaviour of `count_letters(word, letters)` by allowing more than one character in the second argument. Return how many times in total *any* of the given `letters` occurs in `word`. Use two `for` loops, one nested inside the other. Hint: the outer loop should set a variable to each character in `letters`.

Verify that:

```
word = "bananas"
print(count_letters(word, "a"))       prints '3'
print(count_letters(word, "ban"))     prints '6'
print(count_letters(word, "banz"))    prints '6'
print(count_letters(word, "bans"))    prints '7'
```

▷▷ Show your two versions of `find_letter` and three versions of `count_letters` to an instructor **now**.

## 3 Palindromes [2 points]

A *palindrome* is a word that is spelled the same forwards and backwards. (In other words, it is symmetrical about its centre.) Examples include: “kook”, “kayak”, “rotator”, and “deified”.

Write a function `palindrome(string)` that returns `True` if `string` is a palindrome, or `False` if it is not a palindrome. Verify that your function identifies “reviver”, “level”, and “deed” as palindromes, but not “banana” or “lever”.

Hints:

- If `length` is the length of `string` then `string[0]` is its first character and `string[length-1]` is its last character.
- If the variable `i` counts from 0 to `length-1` then `string[i]` iterates through the letters in `string` forwards while `string[length-1-i]` iterates through them backwards.
- If you can iterate through the letters of `string` forwards and backwards, in the same loop, then you can compare each pair of letters to determine if string is a palindrome, or not.

Make a copy of your function called `palindrome1`.

### 3.1 Sequence reversal

The 'slice' operator `s[::-1]` copies the entire string `s` *backwards* (reversing the order of its characters).

Modify `palindrome1(s)` so that it has only *one* single line in the body of the function. Verify it produces the same results as above.

▷▷ Show both versions of `palindrome` to an instructor **now**.

## 4 Challenge [1 bonus point]

A *rotational cypher* is a way to encode a message so that it is difficult to read. It works by rotating (shifting) the letters in the message forward or backward through the alphabet by a certain number of places. (It is called 'rotational' because shifting the letter 'z' forward one position comes back to 'a', and shifting 'a' backward one position comes back to 'z'.) For example, 'abcxyz' rotated by 2 is 'cdezab' and 'abcxyz' rotated by -1 is 'zabwxy'.

Write a function called `rotate(string, n)` that rotates `string` by `n` places. If `n` is positive then it rotates letters forwards towards 'z', and if `n` is negative then it rotates letters backwards towards 'a'. Running `rotate(rotate(string, n), -n)` should return the original string.

Hint: One way to do this is *incremental development* where you write the smallest possible amount of code between testing versions of your function to make sure they behave properly. For example, you might write the function using the following steps:

1. Make a copy of `print_each1(x)` called `rotate(word)` that simply prints each letter in `word`.
2. Modify `rotate(word)` so that instead of printing each letter it prints the letter's *position* in the alphabet. To do this you might use the `ord(c)` function that returns a number corresponding to the letter `c`. To make this relative to the start of the alphabet you could use: '`ord(c) - ord('a')`'. Assume for now that `word` only contains lower case letters.
3. Modify `rotate(word)` so that it adds 1 to the position of each letter in the alphabet before printing it. Ensure that the final 'z' is printed as 0, and not 26. One way to do that is to use the modulus (remainder) operator '%' to make sure the result is always between 0 and 25. Check that the sequence printed for `"pineapplez"` starts with 16, 9, and 14 and that it ends with 12, 5, and 0.
4. Modify `rotate(word)` so that it accepts a second parameter `n`, and rotates each letter's position by `n` instead of 1. (Test with both positive and negative values of `n`.)
5. Modify `rotate(word)` so that it prints each rotated letter as a letter instead of printing its rotated position in the alphabet. To do this you can use the '`chr(n)`' function which prints the character corresponding to position `n`.

6. Modify `rotate(word)` so that it creates a new string from the rotated letters and returns the entire rotated word as its result instead of printing each letter individually. One way to do this is to create an empty string `r` at the start of the function, add each rotated letter to the end of `r`, one at a time, and then return the final value of `r`.
7. Modify `rotate(word, n)` so that it only rotates lower-case letters. One way to check whether a letter `c` is lower case is to use the condition `'a' <= c <= 'z'`. Test your function by rotating `"abc 123"` by 2 places and you should get the result `"cde 123"`.
8. Modify `rotate(word, n)` so that it separately rotates lower-case letters and upper case letters. Test your function by rotating `"This is a message that has been encoded."` 13 places to the right, then decoding the result of that by rotating it -13 places to the left. The final result should be identical to the original message, even though the intermediate result is unreadable.

The version of this cypher with rotation 13 is called 'rot13'. Is there anything special about the number 13? Why do you think 13 was chosen as an especially significant rotation value for this kind of cypher?

▷▷ Ask an instructor to check and record your work **now** (before you run out of time).