The deadline for this exercise sheet is **Monday, 07.05.2018, 18:00.**

# 1 Warm-Up: Regular Expressions

For each of these RegEx, please indicate which of the given statements can be described by the RegEx.
For each of the statements that *cannot* be described by the given RegEx, please explain, why.

## 1.1

Can these statements be described by this RegEx: **ab?c+**

a) abc **Solution:** Yes.

b) ab **Solution:** No. There needs to be at least one c.

c) abcc **Solution:** Yes.

d) accc **Solution:** Yes.

## 1.2

Can these statements be described by this RegEx: **^((xy)|[0-9]*)*$**

a) 159yx1 **Solution:** No. It needs to be xy, not yx.

b) xyxyxyxy1 **Solution:** Yes.

c) 1xy27xy385 **Solution:** Yes.

d) 12345xy67ab89 **Solution:** No. This RegEx does not include the characters a or b.

## 1.3

Can these statements be described by this RegEx: **(x*)(.[a-k])y\2\1**

a) xxxyayyaxxx **Solution:** Yes.

b) xx0my0mxx **Solution:** No. Group 2 consists of some arbitrary character and a letter from a to k. The character is already 0, so we cannot use an m as the second character in this group.

c) #ay#a **Solution:** Yes.

d) xxxefyefxx **Solution:** No. One x is missing at the end.

## 2 DFA

In this task we will look at a simple player controller for a 2D platformer. In this game a player can move to the left and right, jump while standing still and while moving and glide through the air after a jump. The goal the player needs to hit is in the sky and can only be reached by colliding while jumping or gliding. If the player collides while standing or moving, it is game over (the implicit error state).
$U$ means the up key, $L$ the left and $R$ the right key. 0 implies there is no key pressed, and *collide* is whenever the player hits another entity.
Write down the quintuple that defines the finite state automaton $A = (\Sigma, S, S_0, G, \delta)$.

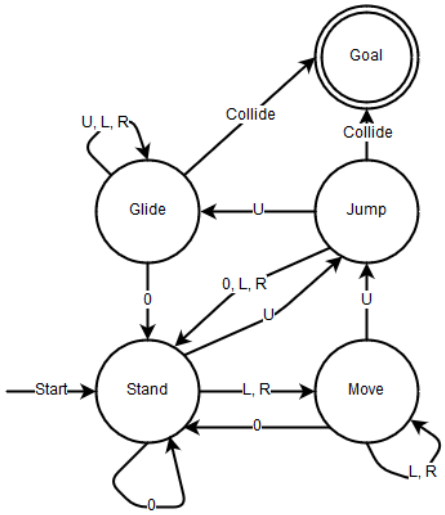


Figure 1: The FSA as a directed graph

**Solution:**
$A = \{\Sigma, S, S_0, G, \delta\}$
$\Sigma = \{U, L, R, U, Collide\}$
$S = \{Stand, Move, Jump, Glide, Goal\}$
$S_0 = Stand$
$G = \{Goal\}$

$\delta =$

| | Stand | Move | Jump | Glide | Goal |
|---|---|---|---|---|---|
| 0 | Stand | Stand | Stand | Stand | ? |
| L | Move | Move | Stand | Glide | ? |
| R | Move | Move | Stand | Glide | ? |
| U | Jump | Jump | Glide | Glide | ? |
| Collide | ? | ? | Goal | Goal | ? |

# 3 Old MacDonald Had A Farm

Most of you will be familiar with the classic nursery rhyme of Old MacDonald and his farm full of animals. The first verse goes like this:

*Old MacDonald had a farm, E-I-E-I-O*
*And on his farm he had a cow, E-I-E-I-O*
*With a moo-moo here and a moo-moo there*
*Here a moo*
*There a moo*
*Everywhere a moo-moo*
*Old MacDonald had a farm, E-I-E-I-O*

In all subsequent verses, the next animal is introduced making its respective sound.
In the file `old_macdonald.py`, you will find a function that reads this first verse out of a file named `old_macdonald.txt`. Open the code and execute the script to look at the output.

## 3.1 Old MacDonald's Cat

Old MacDonald recently also got a cat, which unfortunately could not keep quiet and kept on meowing through the verse, which you will have seen in the output.
Each "meow" starts with either an uppercase or lowercase $M/m$, is followed by at least one $e$, at least one $o$ and ends with exactly one $w$.

Examples:
*Meow*
*meeooow*
*Meeeeeeeeeeow*

The code includes another function `delete_meow`. Right now it does nothing and only returns the same verse as it was given.
Modify the function such that it returns an altered version of the given verse in which each meowing following the exact described pattern is deleted. The function `sub` of the `re` module might be helpful.

**Solution:**

```
1  def delete_meow(verse):
2      """Deletes all the versions of cat noise from the input string.
           """
3      return re.sub("(m|M)(e)+(o)+w",'',verse)
```

## 3.2 Old MacDonald's Whole Farm

Now let's sing the whole song!
First there is the cow going 'moo'.
Then a duck going 'quack'.
Then a horse going 'neigh'.
Then a pig going 'oink'.
And finally the cat, which can now go 'meow' after having waited its turn.
So, for example, the second verse looks like this:

*Old MacDonald had a farm, E-I-E-I-O*
*And on his farm he had a duck, E-I-E-I-O*
*With a quack-quack here and a quack-quack there*
*Here a quack*
*There a quack*
*Everywhere a quack-quack*
*Old MacDonald had a farm, E-I-E-I-O*

Modify the given function `sing_song` to sing the song with all these animals (instead of only printing the first verse, as it does now)!

This is done easiest using string formatting:

- First, find the words "cow" and "moo" in the cat-free verse and replace them with formatting markers
- Now use `.format()` to print the verse once for each animal with its respective sound and an empty line afterwards
- Tip: You can create a compact and flexible solution by making a loop that prints each verse and saving all the animals and sounds in a collection as introduced last week.

**Solution:**

```
1  def sing_song(verse):
2      """
3      Sings the song.
4
5      Replacing all default "moo"s and "cow"s with formatting markers
           to then
6      iterate over a list of animal-sound tuples and fill those markers
           in.
7      """
8      verse_rep = re.sub('cow','{0}', verse)
9      verse_rep = re.sub('moo','{1}', verse_rep)
10     # alternatively to re.sub, we can use
11     # 'verse.replace('cow','{0}').replace('moo','{1}')'
12     # replace might be more suitable since we don't need regex
13
```

```
14      animals = [
15          ('cow','moo'),
16          ('duck','quack'),
17          ('horse','neigh'),
18          ('pig','oink'),
19          ('cat','meow')
20        ]
21
22      for animal in animals:
23          print(verse_rep.format(animal[0],animal[1]),'\n')
```