# Marby's Programming Tutorial for Beginners

# Table of contents

# 1. Preface

This tutorial is written for beginners and does not require any previous knowledge in programming for understanding it. However, it would help you to watch deWiTTERS tutorial video first, before reading it.

So just login on RPG Playground, click “Make” from the menu line, then click “CREATE NEW GAME” and have a look at the tutorial video there. Afterwards, you right can go on by creating your own game and trying the examples from this tutorial.

Programming can be learned only by doing it. Therefore, don't hesitate to try your own ideas and modify the given examples as you like. Change them, test them and see what will happen!

I wish you much fun.

Marby.

# 2. Tokens

If you need to remember things, you can use tokens. When the game starts, the hero possesses not a single token. You can think of a token as of an item the hero has or lacks. This can be something physical like a key of a door as well as something immaterial like an information or an ability.

You can give your hero tokens using commands like the following:

```
hero receives token "Wing of the bat"
hero receives token "Spotted the strange guy"
hero receives token "Able to understand the dwarven language"
hero receives token "Member of the guild"
```

You can name your tokens as you want. However, notice that tokens for the current version of RPG Playground can be used, without to define them. That means, if you do not use the exact name of a token, the token cannot be found by the engine and you will only notice that at runtime. There is no error message yet, which warns you.

```
if hero lacks token "Member Of The Guild"
    Doorman says "Sorry! Only members are allowed to go in."
else
    Doorman says "Welcome to the black market!"
    hero teleports to door "Entrance" at "BlackMarket"
```

Note:
if-statements always tests a condition. The indented program lines beneath the if-statement only gets executed, if the condition is evaluated to be true. Otherwise, the program lines will be skipped and the indented program lines beneath the else-statement will be executed. It is not required to have an else-statement.

Everything in RPG Playground is case sensitive. Whether you're using lowercase or uppercase letters is very important. That goes for everything and also for tokens. Therefore, the tokens `"Member of the guild"` and `"Member Of The Guild"` are completely different tokens. If we make a mistake like this one above, it e. g. means the hero will not even have access to the house as a member of the guild.

# 3. Program flow

## 1. Indentation

Usually your program lines get processed from top to bottom. The following code will bring the words “one” up to “three” to the screen:

```
hero says "one"
hero says "two"
hero says "three"
```

Note:
In programming the commands the computer executes are referred as “code”, “source code” and sometimes as “script”. By referring them as “screenplay” RPG Playground is using its own terminology. Basically all these terms mean the same.

What would you expect as the output assuming the hero lacks the token `"Show also the word two"` with the following screenplay?

```
hero says "one"
if hero has token "Show also the word two"
hero says "two"
hero says "three"
```

Perhaps this will surprise you, but the engine will show all the three messages. This is because none of the program lines above belongs to the if-statement. If you would like lines to belong to that if statement, you've to indent them with four spaces right beneath the if-statement. The screenplay below indeed only shows a message with the word “two”, if the hero has the token “Show also the word two”.

```
hero says "one"
if hero has token "Show also the word two"
    hero says "two"
hero says "three"
```

Is is possible to bind as many program lines as you want to an if-statement.

```
hero says "one"
if hero has token "Show also the word two"
    hero says "two"
    hero says "Two, I said!"
    hero says "Two, two, two, two, two!"
    hero says "Have I already mentioned the number two?"
hero says "three"
```

You can indent your lines more than you need to. The engine even will process it. However, you shouldn't do so. Keep your programs well formatted. The following code will work, but this is not a good programming style.

```
hero receives token "Show also the word two"
hero says "one"
if hero has token "Show also the word two"
        hero says "two"
hero says "three"
```

Note that you will only get rid of the tokens if you restart your game or use a command like the following.

```
hero loses token "Show also the word two"
```

We cannot give our hero the same token multiple times. If the hero already has a particular token, the engine just ignores the “receives token”-command if it would give the hero a token, which already is in the hero's possession. The same goes for the “loses token”-command. If the hero already lacks a token, a “loses token”-command on that token has no effect.

To attain a better readability of your screenplay you can use empty lines in order to format it.

```
hero receives token "Show also the word two"

hero says "one"
if hero has token "Show also the word two"
    hero says "two"
hero says "three"
```

It even is possible to insert empty lines to format the code of an if-statement. The following lines for the word “two” still belongs to the if-statement.

```
hero says "one"
if hero has token "Show also the word two"
    hero says "two"

    hero says "Two, I said!"
    hero says "Two, two, two, two, two!"
    hero says "Have I already mentioned the number two?"
hero says "three"
```

As I mentioned before, it is possible to use a deeper indentation as you need to. However, the other way round will not work. For the following screenplay the editor gives us an error message.

```
hero says "one"
if hero has token "Show also the word two"
    hero says "two"
    hero says "Two, I said!"
hero says "Two, two, two, two, two!"
    hero says "Have I already mentioned the number two?"
hero says "three"
```

Sometimes you need to process command lines for both – if the hero has the token and if the hero don't has the token. This one could be implemented like the following.

```
hero says "one"
if hero has token "Show also the word two"
    hero says "two"
if hero lacks token "Show also the word two"
    hero says "No! I won't say that word!"
hero says "three"
```

Another way to implement it, is to use an else-statement. The program lines, which belongs to an else-statement, will be executed if the condition for its if-statement is evaluated to be false.

```
hero says "one"
if hero has token "Show also the word two"
    hero says "two"
else
    hero says "No! I won't say that word!"
hero says "three"
```

For an else statement exactly the same rules are valid as for an if-statement when it comes to binding of program lines and inserting of additional empty lines.

```
hero says "one"
if hero has token "Show also the word two"
    hero says "two"


    hero says "Two, I said!"
    hero says "Two, two, two, two, two!"

    hero says "Have I already mentioned the number two?"
else


    hero says "No! I won't say that word!"
    hero says "Don't make me to say it!"
hero says "three"
```

An also important thing (as you now might guess) is, that an if-statement as well as an else-statement can remain empty. For both there is no need to bind a program line. Therefore, the output for the following code is: “one”, “three”, “one”, “three”

```
hero receives token "Show also the word two"
hero says "one"
if hero has token "Show also the word two"
else
hero says "three"

hero loses token "Show also the word two"
hero says "one"
if hero has token "Show also the word two"
else
hero says "three"
```

We can nest other if-statements as much as we want beneath an if-statement. Let's have a look at the following example.

```
if hero lacks token "Member of the guild"
    if hero lacks token "Talked to doorman once"
        Doorman says "Sorry! Only members are allowed to go in."
    else
        if hero lacks token "Member card"
            Doorman says "Go away or I'll make you walk!"
    if hero has token "Member card"
        if hero lacks token "Saw the member card"
            hero receives token "Saw the member card"
            Doorman says "What? You have a member card?"
        Doorman says "Go to the boss first and register."
else
    if hero has token "Visited the black market"
        hero receives token "Visited the black market"
        Doorman says "I already know you. Go ahead!"
        hero teleports to door "Entrance" at "BlackMarket"
    else
        if hero has token "Member card"
            hero receives token "Visited the black market"
            Doorman says "Welcome to the black market!"
            Doorman says "You're here the first time. Have fun!"
            hero teleports to door "Entrance" at "BlackMarket"
        else
            if hero lacks token "Talked to doorman once"
                Doorman says "I see your member tattoo."
                Doorman says "Do you have forgotten your card?"
            else
                Doorman says "Or is your tattoo a fake?"

hero receives token "Talked to doorman once"
```

The indentations here are essential. Only with the help of the indentations the engine can tell which command line we would like to bind to which if- or else-statement. Also to identify which if-statement belongs to which else-statement, always have a closer look at the indentation. The if- and else-statements, which belongs together, are always at the same indentation level.

Imagine the entrance in front of the house, in which the black market is in, as a separate place. Every time we're leaving that entrance (the doorman's place), we make sure the following screenplay runs.

```
hero loses token "Talked to doorman once"
```

Now I will explain, what actually happens here. We start our game without the member card and without being a member of the guild. Since we have not talked with the doorman before, he just says, “Sorry! Only members are allowed to go in.”

If we leave this place and return again, without having the member card and without being a member, the doorman will say the same thing again as before, “Sorry! Only members are allowed to go in.”
This is because we made sure to lose the token “Talked to doorman once”, while leaving his place. However, this time we stay and talk again to the doorman. He gets mad at us and says, “Go away or I'll make you walk!”
This will repeat over and over if we have neither a member card, nor being a member of the guild.

Let's say the guild unfortunately does not take more members. However, a former member gives us the hint, he threw away his member card recently behind the house of the black market. So while searching the garbage can there we finally are able to receive such a card. Afterwards, we go back to the doorman. The doorman still notices we are not a guild's member. As before he says, “Sorry! Only members are allowed to go in.” But now, he also sees our member card. He is surprised and adds, “What? You have a member card?” As a last thing he advises us, “Go to the boss first and register.” If we talk to him again afterwards, he is not unfriendly anymore. He just repeats his advice, "Go to the boss first and register."

Of course, his boss never would us give his okay in order to become a member of the guild. However, the hero's friend can make wonderful fake tattoos. Do not bring them in touch with water though. 😉

With such a tattoo and of course the token “Member of the guild” the hero returns. Now the doorman greets us quite friendly. He says “Welcome to the black market!” and “You're here the first time. Have fun!”

If we leave the black market and enter it again the doorman states, “I already know you. Go ahead!”

Our screenplay also works the other way round. If we first visit the hero's friend in order to get the fake tattoo and with it the “Member of the guild”-token, the doorman says afterwards, “I see your member tattoo.” and “Do you have forgotten your card?”
If we talk to him again, he suspects us from now on over and over, “Or is your tattoo a fake?”
Let us now get the member card from the garbage can. If we have it, we will be able to enter the black market like before. This time we just used another way for solving the riddle.

With this example you can see how lively your games will be if you're cleverly using tokens.

## 2. Screenplay after a “teleports to”-command

A “teleports to”-command does not exit the screenplay. After the hero is taken to the other level the next program line will be executed. With the program lines which follows you're not able to access the actors of the other level. Probably this will be possible in the future, but today, as I write this tutorial, it is not yet possible. However, the actors of the previous level remain still accessible until the screenplay ends.

## 3. Error handling

There are two types of mistakes you can make with programming. The first one is regarding the syntax, the last one only has an impact at runtime. If your syntax is okay, you still can have a problem with your program logic. The program works, but not behaves the way you intended, brings a runtime error or even crashes. While sometimes it is pretty hard to find a logic or runtime error, finding a syntax error usually is much easier. Unfortunately, not with RPG Playground yet! At the moment I write this text the editor still will state if there's something wrong with your screenplay's syntax. However, you easily can overlook this, because you still can just click “OK”. The result feels as you have made a logic mistake, because the engine just won't run your screenplay beyond the error.

```
hero says "This will run."
hero says "This also will run."
hero say "Something is wrong with this program line."
hero says "Neither this will run,"
hero says "nor this one!"
```

This behavior might even be useful, but not if we have no list with all errors for every actor and level, which automatically gets updated. At the time I write this here, such a list is missing in RPG Playground.

# 4. Numbers

## 1. Differences of variables and constants

Like tokens also numbers in RPG Playground can be used to store information. In programming languages constructs like tokens and numbers are called “variables”. Therefore, I will also refer to numbers and tokens as variables from now on. Token variables and number variables are very similar, since both of them are able to store information. The difference is, that they have different data types. Unlike token variables, which store if something is there or not, with numbers variables you are able to store numbers of your choice.

In order not to mix up things, you should know, that there is a difference between number variables and number constants. Therefore, I personally don't like to call number variables just numbers. Next we will have a look at what the difference is.

Here are some number constants:

- `0`
- `-10`
- `55`
- `-0.2`
- `3.1415`

Number constants cannot change their values. They always have the value they have.

And here are some number variables:

- `hero number of "ingredients"`
- `hero number of "current level"`
- `hero number of "puzzle step"`
- `hero number of "gold coins"`
- `hero number of "calculation result"`

You can change the value of number variables at runtime.

There is one thing you probably already have noticed. We always have to write “hero number of” in order to refer to a number variable. However, the number variables we define have not necessarily something to do with our hero. Don't get confused by this. Just use your number variables the way you want. For tokens this of course is similar. It's always up to you how you use tokens in your game.

As for token variables before and all the other identifiers in RPG Playground, also number variables are case sensitive. Therefore, the number variables “gold coins”, “Gold Coins” and “GOLD COINS” are completely different variables.

Do you remember there is no need to define token variables, before you can use them? The same goes for number variables. If you check the content of a number variable you have not used before, it will have the value 0.

## 2. Manipulation of number variables

It is possible to directly set the value of a number variable to a value of a number constant. The following statement will set the content of a number variable named “gold coins” to 10. In programming such a statement is called an assignment.

```
hero number of "gold coins" becomes 10
```

You can increase the value of a number variable with the “increases with”-operator.

```
hero number of "gold coins" increases with 5
```

After running the both lines of our example here, the value of number variable “gold coins” is 15. Let's subtract 3 gold coins in order to have 12 gold coins left. This is possible with the “decreases with”-operator.

```
hero number of "gold coins" decreases with 3
```

As I write this tutorial, there only are the three possibilities, I just mentioned, to change the values of number variables and there is not yet a way to send their values directly to a screen message. In order to check if our number variables have the correct values, we for now have to work with if-statements, as you will understand later on.

## 3. Comparisons

For comparing number variables with number constants there are five operators available.

With the equality operator you can test a number variable for an exact content.

```
if hero number of "gold coins" = 5
    hero says "I have exactly the money I need, in order to buy this."
```

The “less than” operator let you test, if the value of a number variable is less than a number constant.

```
if hero number of "gold coins" < 5
    hero says "I have not yet enough money."
```

Of course, there is also the “greater than” operator to test the other way round. The following code will test if the hero has more than five gold coins.

```
if hero number of "gold coins" > 5
    hero says "I have more gold coins than it costs."
```

Up to now there are two more operators you also can use for comparisons.

- <=
  less than or equal to
  The condition is true, if the left hand operand is less than or equal to the right hand operand.
- >=
  greater than or equal to
  This operator works the other way round. The left hand operand has to be greater than or equal to the right hand operand in order to let the condition be true.

Let us now consider an example to illustrate this. In this example, the hero has to collect some glowworms in a lamp in order to have enough light for entering a cave.

```
hero loses token "Message has been shown"
if hero number of "glowworms" < 5
    LampSalesman says "With so few glowworms, you're in the dark. You need much more!"
    hero receives token "Message has been shown"

if hero lacks token "Message has been shown"
    if hero number of "glowworms" < 10
        LampSalesman says "Looks quite good already. But you still need a few glowworms."
        hero receives token "Message has been shown"

if hero lacks token "Message has been shown"
    if hero number of "glowworms" < 15
        LampSalesman says "You need only a few more glowworms. You've almost made it."
    else
        LampSalesman says "Your lamp is now bright enough to go into the cave!"
```

Without using a token like “Message has been shown” we would get multiple messages. Let's say we have not a single glowworm. That would mean we have less than five, less then ten and less than fifteen glowworms and three messages would pop up. The token “Message has been shown” prevents this. First we need to remove it in order to initialize our screenplay. This is necessary to even process the screenplay properly if the hero talks to the LampSalesman several times. Otherwise, with more then four glowworms no messages at all would pop up. As soon as our program determines the correct message, the message will be displayed, the token “Message has been shown” will be set and therefore further messages are prevented. For the last message block there is no need to set the token, because there are no more messages which possibly could pop up.

# 4. Properties

There are many properties as well for the hero as also for our other actors. Some of them contain number values and behave like number variables too. The values of those properties can be changed at runtime. Here are some examples:

- `hero health value`
- `hero health max`
- `hero walk ability max speed`
- `hero swing weapon ability damage`
- `hero swing weapon ability cooldown time`

If the “Insert action”-button does not provide what you're looking for, just have a look at the property lists of your actors. That way you can figure out what to enter in order to access a particular property.

## 1. Manipulation of properties

By changing properties you can give your hero super powers, change the walking speed of actors or even give your monsters healing forces, which are able to heal instead of hurting the hero. Of course, you also can heal your monsters.

```
hero swing weapon ability damage becomes 100
hero swing weapon ability cooldown time becomes 0
hero walk ability max speed increases with 1
BigBull bull rush ability damage becomes -10
BigBull health value increases with 1000
```

As you can see, the number properties can be changed the same way we also can use to change number variables.

The last line of this screenplay is worth a look. It will restore 1000 HP to the BigBull's health, will it? Or will it not? The answer of this question actually depends on how high the maximum health of the monster is. If the new value would exceed the maximum health, the HP will be replenished only up to the maximum health.

## 2. Comparison of properties

All the comparison operators for number variables also work with number properties. Let's define the following screenplay using the “When hurt”-event for one of our monsters in order to demonstrate it.

```
if hero lacks token "holy sword"
    if BigBull health value <= 0
        BigBull health value becomes 200
        BigBullCopy says "He he! You cannot kill me with that pathetic sword of yours!"
```

As you can see, the screenplay above compares the health value of our BigBull monster with zero. If it is less than or equal to zero, the monster is dead. A hit actually can reduce the hit points below zero. Therefore, it indeed is necessary to check for less than or equal to zero.

In order to restore its health we have to check in the “when hurt”-event, if the current hit killed the monster. If it was not killed with the holy sword, it recovers.

Note:
There has been an issue with the “When hurt”-event. If your screenplay doesn't work, restart your game and try it again. It is possible that changed screenplay only gets applied to a “When hurt”-event after a restart.

It also is possible, that a particular property you want to use is not supported by the actor type you chose. In that case you should replace that actor to another one who can handle it. Since we're able to let our actors look the way we want, this approach is a good workaround.

However, for our BigBull this does not work for now. This is because we want it to speak as well as fight. The sprite of a bull is unable to use a sword though as well as a human cannot use the bull rush ability. Therefore, we've got the problem to either let our monster fight or let it talk. As a solution for this problem I inserted a human actor, turned that actor into a bull named “BigBullCopy” and than let the bull talk as usual. You should set your “copy”-actors to inactive in order to remove them from your game. This can be done with the following program line.

```
BigBullCopy becomes inactive
```

Nevertheless, you can even use the says-command on inactive characters to let them speak.

As I write this, there are no “and”- or “or”-operators in RPG Playground that we could use to build more complex conditions. Therefore, we need to use if-statements as I did in the previous example to make our screenplay work as if we already had an “and”-operator.

# 5. Compatibility

There unfortunately are some limitations yet. It is for example not possible to assign the value of a number variable or property to another number variable or property. The same goes for comparisons. The following statement will not work properly:

```
hero number of "coins in backpack" becomes hero number of "gold coins"
```

deWiTTERS already stated that we will soon have the capability to use our variables like this. However, currently an invalid value will be assigned to our variable, which is obviously not even a number. Let's have a look at the following screenplay for a better understanding.

```
hero number of "coins in backpack" becomes 10
hero number of "gold coins" becomes 123
hero number of "coins in backpack" becomes hero number of "gold coins"

if hero number of "coins in backpack" = 123
    hero says "I've 123 gold coins in my backpack."
else
    hero says "Where are my gold coins?"

if hero number of "coins in backpack" = 0
    hero says "I've 0 gold coins in my backpack."
else
    hero says "At least I've not 0 gold coins in my backpack."

if hero number of "coins in backpack" < 0
    hero says "I've a negative number of gold coins in my backpack."
else
    hero says "At least I don't have any debts."

if hero number of "coins in backpack" > 0
    hero says "Okay! There is something of value in my backpack."
else
    hero says "I searched the whole backpack and haven't found a single gold coin."

hero says "Better I only trust number constants."
hero number of "coins in backpack" becomes 15

if hero number of "coins in backpack" = 15
    hero says "Yey! Number constants are cool!"
else
    hero says "What the...? I better stay home for today!"
```

If we run this we'll get the following messages:

- "Where are my gold coins?"
- "At least I've not 0 gold coins in my backpack."
- "At least I don't have any debts."
- "I searched the whole backpack and haven't found a single gold coin."
- "Better I only trust number constants."
- "Yey! Number constants are cool!"

A valid number is either greater than zero, or equals to zero, or is less than zero. In our case the messages states, that the value of our number variable “coins in backpack” after the assignment with the value of “gold coins” is none of them. That means, that its value now is not a number anymore. Therefore be warned. You cannot use it that way until this bug has been fixed.

# 5. What we can do with variables

There are lots of possibilities what you can do with token and number variables. The limitation is only your imagination. I will show two examples in order to give you a hunch of what you are capable of if you understood how to use variables.

## a. Example 1 – Collecting and checking ingredients

Imagine a witch, who needs a whole bunch of ingredients in order to make a potion for rescuing a fairy.

```
if hero lacks token "potion of the witch"
    if hero has token "wing of the bat"
        Witch says "Oh! You have the wing of a bat!"
        if hero has token "tooth of the snake"
            Witch says "I see, my dear. A snake's tooth is in your possession."
            if hero has token "egg of the lizard"
                Witch says "Yes, yes, yes... such a lizard's egg will come in handy."
                if hero has token "nail of the mouse"
                    Witch says "Very good. Finding a mouse's nail surely was no easy task."
                    Witch says "Very well, my dear! Here is your potion!"
                    hero receives token "potion of the witch"
                else
                    Witch says "No, my dear!"
                    Witch says "Unfortunately, you haven't all ingredients yet."
            else
                Witch says "No, my dear!"
                Witch says "Unfortunately, you haven't all ingredients yet."
        else
            Witch says "No, my dear!"
            Witch says "Unfortunately, you haven't all ingredients yet."
    else
        Witch says "No, my dear!"
        Witch says "Unfortunately, you haven't all ingredients yet."
else
    Witch says "Now go, my dear. I pretty much hope it will help your fairy friend."
```

This approach even has multiple issues.

1. You have to indent this insanely deep. This will make your code hard to read and hard to maintain. Imagine you've even more than only four ingredients. At some point it is not fun anymore.
2. You have to repeat code over and over for the case of missing ingredients (else-statement). If you want to change what the witch is saying, you have to do it multiple times.
3. The witch also is telling you, what you've already collected so far. However, imagine you get the “wing of the bat” at the very end. In that case she will never say something about your progression until you finished the complete task. In other cases she often will not mention ingredients you also already found.

Let's try to make this better.

```
hero loses token "some ingredients are missing"

if hero lacks token "potion of the witch"
    if hero has token "wing of the bat"
        Witch says "Oh! You have the wing of a bat!"
    if hero has token "tooth of the snake"
        Witch says "I see, my dear. A snake's tooth is in your possession."
    if hero has token "egg of the lizard"
        Witch says "Yes, yes, yes... such a lizard's egg will come in handy."
    if hero has token "nail of the mouse"
        Witch says "Very good. Finding a mouse's nail surely was no easy task."

    if hero has token "wing of the bat"
        if hero has token "tooth of the snake"
            if hero has token "egg of the lizard"
                if hero has token "nail of the mouse"
                    Witch says "Very well, my dear! Here is your potion!"
                    hero receives token "potion of the witch"
                else
                    hero receives token "some ingredients are missing"
            else
                hero receives token "some ingredients are missing"
        else
            hero receives token "some ingredients are missing"
    else
        hero receives token "some ingredients are missing"

    if hero has token "some ingredients are missing"
        Witch says "No, my dear!"
        Witch says "Unfortunately, you haven't all ingredients yet."
else
    Witch says "Now go, my dear. I pretty much hope it will help your fairy friend."
```

Two of the three issues has been resolved with this.

1. As you can see, you still have to indent your programming lines insanely deep.
2. However, you don't have to repeat the code anymore, which lets the witch say that ingredients are missing. Now, if you would like to change something about that, you only have to do it in one place.
3. The order you collecting the ingredients also doesn't matter anymore. Every time you talk to her, the witch will tell you all the ingredients you've collected so far, without an exception.

If we use number variables we can improve this screenplay even more.

```
hero number of "collected ingredients" becomes 0

if hero lacks token "potion of the witch"
    if hero has token "wing of the bat"
        Witch says "Oh! You have the wing of a bat!"
        hero number of "collected ingredients" increases with 1
    if hero has token "tooth of the snake"
        Witch says "I see, my dear. A snake's tooth is in your possession."
        hero number of "collected ingredients" increases with 1
    if hero has token "egg of the lizard"
        Witch says "Yes, yes, yes... such a lizard's egg will come in handy."
        hero number of "collected ingredients" increases with 1
    if hero has token "nail of the mouse"
        Witch says "Very good. Finding a mouse's nail surely was no easy task."
        hero number of "collected ingredients" increases with 1

    if hero number of "collected ingredients" = 4
        Witch says "Very well, my dear! Here is your potion!"
        hero receives token "potion of the witch"
    else
        Witch says "No, my dear!"
        Witch says "Unfortunately, you haven't all ingredients yet."
else
    Witch says "Now go, my dear. I pretty much hope it will help your fairy friend."
```

Now also the deep indentations are gone. As you can see, number variables can help you a lot to improve your screenplay. Now even with 20 ingredients, implementing this is not a big task for you anymore and even then your code will remain readable and good to maintain.

## b. Example 2 – Handle messages of an actor

Usually, it is better to let your actors not to say too much at once. Have a look at the following example.

```
OldBill says "Do you see the dog over here? His name is Junior. If you ask me, Junior is behaving strange lately. He's calm all the time... almost majestic. He didn't used to be like that. Just take a look in his eyes. Don't they look strangely knowing? People already think I'm crazy. Do you also think so? You simply have no respect for age anymore."
```

Do you really think players would enjoy to read that much text at once? Imagine a game, where every actor has so much to say. If you play this, I bet you rather will skip the messages at some point instead of reading them. To make things worse OldBill even repeats himself over and over! That will delay the game and also be annoying if the player accidentally talks to OldBill again. As you can see, this is not a good style of game design.

```
if hero lacks token "Talked to OldBill"
    hero receives token "Talked to OldBill"
    OldBill says "Do you see the dog over here? His name is Junior."
else
    if hero lacks token "Talked to OldBill twice"
        hero receives token "Talked to OldBill twice"
        OldBill says "If you ask me, Junior is behaving strange lately."
    else
        if hero lacks token "Talked to OldBill three times"
            hero receives token "Talked to OldBill three times"
            OldBill says "He's calm all the time... almost majestic."
            OldBill says "He didn't used to be like that."
        else
            if hero lacks token "Talked to OldBill four times"
                hero receives token "Talked to OldBill four times"
                OldBill says "Just take a look in his eyes. Don't they look strangely knowing?"
            else
                if hero lacks token "Talked to OldBill five times"
                    hero receives token "Talked to OldBill five times"
                    OldBill says "People already think I'm crazy. Do you also think so?"
                else
                    OldBill says "You simply have no respect for age anymore."
```

Now the gameplay has become a lot more fluently. Also OldBill is more interesting now, because he is not talking the same thing all the time anymore and the player can decide on how much to talk to OldBill. We can program this that way, if an actor has not that much to tell. However, OldBill has to tell a lot! See how deeply we need to indent our program lines for realizing it this way. If we need to get rid of all the tokens, because the hero leaves the town and we want OldBill to start his talk from the beginning next time, we have to write a whole bunch of program lines.

```
hero loses token "Talked to OldBill"
hero loses token "Talked to OldBill twice"
hero loses token "Talked to OldBill three times"
hero loses token "Talked to OldBill four times"
hero loses token "Talked to OldBill five times"
```

Therefore, in this case it is way better to use a number variable instead of token variables.

```
if hero number of "Spoken with OldBill" = 0
    OldBill says "Do you see the dog over here? His name is Junior."
if hero number of "Spoken with OldBill" = 1
    OldBill says "If you ask me, Junior is behaving strange lately."
if hero number of "Spoken with OldBill" = 2
    OldBill says "He's calm all the time... almost majestic."
    OldBill says "He didn't used to be like that."
if hero number of "Spoken with OldBill" = 3
    OldBill says "Just take a look in his eyes. Don't they look strangely knowing?"
if hero number of "Spoken with OldBill" = 4
    OldBill says "People already think I'm crazy. Do you also think so?"
if hero number of "Spoken with OldBill" = 5
    OldBill says "You simply have no respect for age anymore."

if hero number of "Spoken with OldBill" < 5
    hero number of "Spoken with OldBill" increases with 1
```

To the player there is no difference. However, the structure of our screenplay is much more readable and expandable than before. That way you easily can do even things like the following.

Let's say we reset the number variable “Spoken with OldBill” to zero every time we leave the village. On our way to the fields we meet a young woman named Aneka who claims to be able to read the minds of animals. If we return to the village afterwards, the situation has changed. Also OldBill now is telling us completely different things than before.

```
if hero number of "Spoken with OldBill" = 0
    if hero has token "Met Aneka"
        hero number of "Spoken with OldBill" becomes 6

if hero number of "Spoken with OldBill" = 0
    OldBill says "Do you see the dog over here? His name is Junior."
if hero number of "Spoken with OldBill" = 1
    OldBill says "If you ask me, Junior is behaving strange lately."
if hero number of "Spoken with OldBill" = 2
    OldBill says "He's calm all the time... almost majestic. He didn't used to be like that."
if hero number of "Spoken with OldBill" = 3
    OldBill says "Just take a look in his eyes. Don't they look strangely knowing?"
if hero number of "Spoken with OldBill" = 4
    OldBill says "People already think I'm crazy. Do you also think so?"
if hero number of "Spoken with OldBill" = 5
    OldBill says "You simply have no respect for age anymore."

if hero number of "Spoken with OldBill" = 6
    OldBill says "This woman said that our junior can sniff out gold."
if hero number of "Spoken with OldBill" = 7
    OldBill says "A whole crowd of people has stormed our village since then."
if hero number of "Spoken with OldBill" = 8
    OldBill says "How can they possibly believe a dog can do such a thing?"
if hero number of "Spoken with OldBill" = 9
    OldBill says "Just look at all these silly people."

if hero lacks token "Met Aneka"
    if hero number of "Spoken with OldBill" < 5
        hero number of "Spoken with OldBill" increases with 1
else
    if hero number of "Spoken with OldBill" < 9
        hero number of "Spoken with OldBill" increases with 1
```

The example above requires to start with zero as the value for our number variable “Spoken with OldBill”. This is the case every time you restart the game. You have to make sure to reset it to zero at least, after you have spoken with Aneka.

Let's have a closer look to the screenplay. It consists of four programming blocks. The first block of the screenplay will notice, if you already met Aneka and initializes “Spoken with OldBill” accordingly. Let's call it “initializer”.

After the initializer has run, the screenplay evaluates the correct message to show, almost like before. However, this time there are two blocks of screenplay containing messages instead of one. You even can remove the empty line between message five and six. It will not effect the execution of the screenplay. I only inserted it to clearly show you, which messages belong together.

The last block causes the value of “Spoken with OldBill” to be increased depending on whether you met Aneka or not. If you haven't met her yet, it will be increased up to five, otherwise up to nine. Therefore, when you start the game, the sentence of OldBill that will be repeated over and over, after you spoke to him several times, is, “You simply have no respect for age anymore.” and after meeting Aneka it is, “Just look at all these silly people.”