Python Programming

Beginners Workshop

The „Rock, Paper, Scissors“ Project

Writing the first Program

When introducing beginners to a new programming language, it is difficult to find a balance between too much theory in the beginning and to overburden the candidates with pactical tasks, they are not yet ready to solve.

Too difficult, obviously 😒

Just opening a big toolbox and showing dozens of different tools is not enough. There must be a step-by-step introduction, how to create a first working program from a pure textual description of the task.

So, here the task again:

Rock, Paper, Scissor

Write a Program to play RPS with the computer

- The rules are simple:
- Rock wins against Scissor (makes it 'unsharp')
- Scissor wins against Paper (cuts it)
- Paper wins against Rock (wraps it)
- Equal bets are draws, no win points
- The computer makes an internal bet on R, P or S
- The user enters his bet
- The computer shows the result. (my bet, your bet, I win, you win, draw)
- The computer counts the win points.
- Who first gets 10 win points, wins the game

Some Translation needed

From a description to a 'specification'

A description of a program hardly reflects the steps, which a program must take. The mind of the writer may jump around freely. A program can not.

Take the last sentence as an example:

- Who first gets 10 win points, wins the game

Only then we find out, that the task requires a **repetition** of a single game, requiring a number of wins to finish the match.

Also at the end of the description:

- The computer counts the win points.

To count we need a counter, that is a variable with a numeric value. We obviously need at least two counters. Counters (as most variables) need some **initializing**, before they can be used.

From a description to a 'specification'

The next phrase:

- The computer shows the result. (my bet, your bet, I win ...)

There is some text output from the program, talking to the user. The phrase only mentions the result of a single game, but it should be obvious, that the user also needs some introduction message, and a message about the final result of the match.

Then:

- The computer makes an internal bet on R, P or S

The game called „Rocks, Papers and Scissors“, but when humans play this game, they use a fist, a flat hand and spread fingers to represent the three items. Very often programming is about finding a good **representation** of real world objects.

As a shortcut: let's take the three letters 'r', 'p' and 's'.

From a description to a 'specification'

A bet made by the computer sounds strange, as a computer has no mind. But it would be enough to **simulate** a behavior, which can not be predicted by the player. For this a program can use some form of random generators.

One central phrase:

- The user enters his bet

When a user comes into play, an alarm should go off. Handling **user input** requires special attention. Users behave unpredictably and a program must be aware of all kind of input, which a user might provide.

So this usually requires extended checking of values and also an extra loop, to allow the user to repeat his/her input in cases, when something went wrong.

From a description to a 'specification'

Finally:

The rules are simple: ….

What the description of the game mentioned first: „The Rules“, is actually what comes last in the logic of our program. After we have found a representation of the game ojects, after we have the bets of the computer and the player, we can now apply the rules.

The actual writing if a program is called **implementation**: Transforming ideas into the code of a specific program language.

The implementation of the rules is, what comes closest to our ideas about programming: Checking values, doing comparisons and calculations, making decisons and finaly come out with a result.

There is usually **more than one way** to implement a solution. We will see different approaches soon, learning about some powerful features of the python language.

Before we start ...

Some recommendations

- Programs should be written for humans
- Names of variables should be meaningful
- Comments may be useful
- Program code, which has an obvious meaning is better

There are more rules to follow, but for now: let's start!

The first statements

- Open the IDLE editor and type the first line:

```
# python3
```

This indicates the language and the version of Python (as opposed to version 2)

- Save the file under a meaningful name.

- Write a string, which (shortly) describes the program.

```
''' Rock, Paper, Scissor – a small computer game
    Written by Tom Dewly, 2016
'''
```

A String, which appears ouside any statement, is treated as a comment.

- Import the random module:

```
import random
```

Random functions allow to simulate unpredictable behaviour

Initialize some Variables

- Specify the numer of points to win a match

```
match_points = 10
```

This way, we can later change the number of rounds easily, e.g. for testing.
One more rule applies here: Don't use „magic numbers“

- Initialize two counters.

```
comp_points = 0
user_points = 0
```

- Or make it more like „pythonic“

```
comp_points = user_points = 0
```

Its completely legal to use features of Python like this. It is nothing „tricky“ here.

- Define the valid bets for the comp and the user

```
valid_bets = 'rps'
```

A simple string containing the valid letters should be sufficient

More initializing

- More initialization steps will be included here later …

- Greet the user

```
msg = '''Hello User - let's play 'Rock, Paper, Scissor'
Who first gets {} points, wins the match.
Let's start!'''
print(msg.format(match_points))
```

.

Define the Loop

- Lets use a while loop with a condition. The loop is entered and repeated only, when the condition is true.

```
while user_points < match_points and comp_points < match_points:
```

This following statements need to be indented, but this presentation can not show the indentation correctly.

- Get a bet for the computer. We use one method of the random module: random.choice(list). This method selects one random element of the list given as an argument. It is as simple as this:

```
comp_bet = random.choice(valid_bets)
```

A string can be considered as a list of single letters, so it is a valid parameter to the choice method

- Now let's get the bet from the user, first define the message:

```
ask_user = 'Enter your bet: r, p or s. Enter x to end the game '
```

Its perhaps a good idea, to give the user a choice to leave the game

Input from the User

- Lets use a while loop with a condition. First set the response, then enter the loop:

```
response = ''
while response == '':
    response = input(ask_user)
```

- Now let's check the user Input:

```
    if response == 'x':
        break
    if response in valid_bets:
        break
```

As soon, as we can find a valid answer, we leave the input loop. We could also write:

```
    if response in valid_bets or response == 'x':
        break
```

- All other responses are invalid: let the user repeat its input

```
    print('wrong input, please try again ...'
    response = ''    # stay in the input loop
```

Checking the bets

- Now we left the input loop, but we are still in the game loop.We check the response again

```
if response == 'x':
    print('you give up? - ok, see you soon')
    break
else:
    user_bet = reponse
```

The break leaves the game loop, else we have both bets

- Handle the 'draw' condition first:

```
if user_bet == comp_bet:
    print('your bet was {}, my bet was {}, that\'s a draw')
    continue
```

The 'continue' returns to the start of the Game loop:

- Now we have two valid bets, let's find out who wins

Determine the Winner

- One possible solution would be:

```
if comp_bet == 'r' and user_bet == 's':
    winner = 'comp'
elif comp_bet == 's' and user_bet == 'p':
    winner = 'comp'
elif comp_bet == 'p' and user_bet == 'r':
    winner = 'comp'
else:
    winner = 'user'
```

This looks clumsy? -It is! There must be better ways:

- This one is shorter (and shorter often means: easier to understand)

```
if ( comp_bet == 'r' and user_bet == 's' or
     comp_bet == 's' and user_bet == 'p' or
     comp_bet == 'p' and user_bet == 'r' ):
    winner = 'comp'
else:
    winner = 'user'
```

better already ... - we need to use parenthesis around the condition, so we can write it in more than one line

Determine the Winner

- We could compare tuples of values:

```
if ( (comp_bet, user_bet) == ('r','s') or
     (comp_bet, user_bet) == ('s','p') or
     (comp_bet, user_bet) == ('p','r') ) :
    winner = 'comp'
else:
    winner = 'user'
```

Not much better ... (again watch the parenthesis)

- Lets use an elegant solution, which is possible in Python

```
if comp_bet + user_bet in 'rs sp pr'.split():
    winner = 'comp'
else:
    winner = 'user'
```

ok, we could leave it like this

Process the Result of one Game

- Prepare the user message

```
result_msg = ('Your bet: {}, my bet: {}, {}  - '
              'your points: {}, my points:{}' )
```

watch the parenthesis again: inside parenthesis a long string may be split up)

- Now comes the grand final

```
    if winner == 'comp':
        who_wins = 'I win'
        comp_points += 1
    else:
        who_wins = 'you win'
        user_points += 1
    print(result_msg.format(user_bet, comp_bet, who_wins,
                            user_points, comp_points))
```

This was the last statement inside the while loop

Final Result of the Match

- Now we are outside the loop (no more indentation). We will give a last message for the result of the match

```
if user_points >= match_points:
    print('You won, sure that was pure luck, pah!')
if comp_points >= match_points:
    print('I won! - hey, I am smarter than you, loooser!')
```

Attention: resist the idea to use the 'else' branch for the computer winner. Remember – we leave the loop with a break, when the user enters an 'x'! Then both counters would be below the match_points value.

The program is complete and should run. Try it, test it, experiment!

I would recommend to type the entire program instead of copying the statements. Its a good exercise. And the text in the presentation contains tabulators, which must be changed to spaces manually

fablab
LISBOA

python™