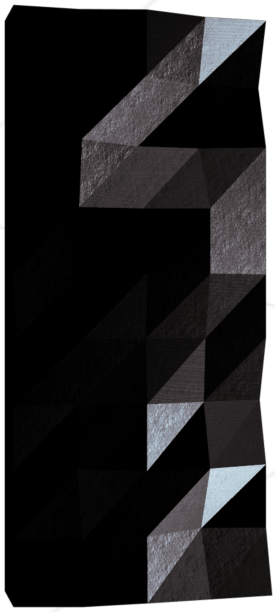

Rush

Summary: This document is the subject for Rush00 of the Python Discovery Piscine @42Bangkok.

armel - armel@42bangkok.com
krittin - krittin@42bangkok.com

Version: 1.1

Contents

Chapter I

The Instructions

- Each member of the group can register the whole group to defense.
- The group MUST be registered to defense.
- Any question concerning the subject would complicate the subject
- You have to follow the submission procedures for all your exercises.
- You have to follow the submission procedures for all your exercises.
- You will have to handle errors coherently. Feel free to either print an error message, or simply return control to the user.
- Rushes exercises have to be carried out in group of 2.
- You must therefore do the project with the imposed team and show up at Your defense slot, with all of your teammates.
- You project must be done by the time you get to defense. The purpose of defense is for you to present and explain your work.
- Each member of your group must be fully aware of the works of the project. Should you choose to split the workload, make sure you all understand what everybody's done. During the defense, you'll be asked questions and the final grade will be based on the worst explanations.
- It goes without saying, but gathering the group is your responsibility. You've got all the means to get in contact with your teammates: phone, email, carrier pigeon, spiritism, etc. So don't bother blurping up excuses. Life isn't fair, that's just the way it is.
- However, if you've really tried everything one of your teammates remains unreachable: do the project anyway, and we'll try and see what we can do about it during the defense. Even if the group leader is missing, you still have access to the submission directory.
- You must use the version of Python announced during the Piscine.

Make sure the subject that was originally assigned to your group works perfectly before considering bonuses: If a bonus subject works, but the original one fails the tests, you'll get 0.

Chapter II

Rush00

	Exercise : 00
Rush00	
Turn-in directory : $ex00/$	
Files to turn in : `main.py checkmate.py`	
Forbidden functions : `None`	

Write a function that takes rows of a chessboard as arguments and checks if your King is "in check".

- Quick reminder, chess is a game played on a chessboard, an 8 smaller-square long square board with specific pieces: King, Queen, Bishop, Knight, Rook and Pawns. For this exercice, you will only play with Pawns, Bishops, Rooks, Queens... and a King.
- A piece can only capture the first possible piece that stands on its path.
- The board can be of different sizes but will remain a square. There's only one King and all other pieces are against it. All the characters that are not used to refer to pieces are considered as empty squares.
- The King is considered to be "in check" when an other enemy piece can capture it. When it's the case, you will print "Success" on the standard output followed by a newline, otherwise you will print "Fail" followed by a newline.
- Your function should never crash or loop indefinitely.
- Whenever an undefiend behavior occurs you should return an error message or prints nothing and gives back control.
- Your main will be modified during defense, to check if you've handled everything you're supposed to. Here's an example of test we'll perform:

Example1)

```
def main():
    board = """\
R...
.K..
..P.
....\
"""
    checkmate(board)

if __name__ == "__main__":
    main()
```

Example2)

```
def main():
    board = """\
..
.K\
"""
    checkmate(board)

if __name__ == "__main__":
    main()
```

- Here's an example of how we run the test.

```
$> python3 main.py | cat -e
Success$
```

- Each piece has specific moves available, and all patterns to capture enemy pieces are detailed below.

```
Pawn (P):
. . . . . . .
. . . . . . .
. . X . X . .
. . . P . . .
. . . . . . .
. . . . . . .
. . . . . . .

Bishop (B):
X . . . . . X
. X . . . X .
. . X . X . .
. . . B . . .
. . X . X . .
. X . . . X .
X . . . . . X

Rook (R):
. . . X . . .
. . . X . . .
. . . X . . .
X X X R X X X
. . . X . . .
. . . X . . .
. . . X . . .

Queen (Q)
X . . X . . X
. X . X . X .
. . X X X . .
X X X Q X X X
. . X X X . .
. X . X . X .
X . . X . . X
```

Chapter III

Bonus

	Exercise : 01
Bonus	
Turn-in directory : $ex01/$	
Files to turn in : *	
Forbidden functions : None	

Add more functionality to ex00. The bonus has 2 parts to be completed.

For the first part, you can choose "one" of the 3 options, it will attain you 15 bonus marks.

- Create a program that accept files containing rows of a chessboard as arguments and checks if your King is "in check".

```
$> python3 main.py valid_board.chess | cat -e
Success$
$> python3 main.py valid_board.chess valid_board2.chess | cat -e
Success$
Success$
$> python3 main.py invalid_board.chess | cat -e
Error$
$> python3 main.py invalid_board.chess valid_board2.chess | cat -e
Error$
Success$
```

Here's an example of possible valid_board.chess:

```
R...
.K..
..P.
....
```

Here's an example of possible invalid_board.chess:

```
R...
.K.....
..P.
....
```

```
R...
.K..
```

```
..K.
....
```

```
R...
....
..P.
....
```

- Create a program that takes the best move for a checkmate.
- Create a chessgame.

For the second part, you can choose to create a (creative) functionality, you will be attained 10 bonus marks.

You will be given the maximum of 25 bonus marks. Bonus will be graded, and only graded, when the mandatory part of the subject is perfect.

Your creative part of the bonus doesn't have to be the same as the examples. You can be creative and add any functionality you want, as long as you justify it.