Writing Simple Games in Python

Games are a terrific way to learn. If you take something simple you know well, you have all the information you need to complete it. Something simple like tic-tac-toe – you know you need a board, some way for the user to select a cell, you need to keep track of who's playing (X or O), when they've made a bad move, and when someone has won. Games often need random values, interact with the user, employ infinite loops – in short, they are fascinating and fun to program and play.

Guessing Game

Let's write a simple program where the user has to guess a random number.

```
$ ./guess.py
[0] Guess a number between 1 and 50 (q to quit): 25
You guessed "25"
Too high.
[1] Guess a number between 1 and 50 (q to quit): 12
You guessed "12"
Too low.
[2] Guess a number between 1 and 50 (q to quit): 20
You guessed "20"
Too high.
[3] Guess a number between 1 and 50 (q to quit): 17
You guessed "17"
Too low.
[4] Guess a number between 1 and 50 (q to quit): 18
You guessed "18"
Too many guesses! The number was "19."
```

To start, we'll use the "new_py.py" script to stub out the boilerplate. I'll use the `-a` flag to indicate that I want the program to use the `argparse` module so we can accept some named arguments to our script:

```
$ new_py.py -a guess
Done, see new script "guess.py."
$ cat -n guess.py
     1	#!/usr/bin/env python3
     2	"""docstring"""
     3
     4	import argparse
     5	import sys
     6
     7	# --------------------------------------------------
```

```
def get_args():
    """get args"""
    parser = argparse.ArgumentParser(description='Argparse Python script')
    parser.add_argument('positional', metavar='str', help='A positional argument')
    parser.add_argument('-a', '--arg', help='A named string argument',
                        metavar='str', type=str, default='')
    parser.add_argument('-i', '--int', help='A named integer argument',
                        metavar='int', type=int, default=0)
    parser.add_argument('-f', '--flag', help='A boolean flag',
                        action='store_true')
    return parser.parse_args()

# --------------------------------------------------
def main():
    """main"""
    args = get_args()
    str_arg = args.arg
    int_arg = args.int
    flag_arg = args.flag
    pos_arg = args.positional

    print('str_arg = "{}"'.format(str_arg))
    print('int_arg = "{}"'.format(int_arg))
    print('flag_arg = "{}"'.format(flag_arg))
    print('positional = "{}"'.format(pos_arg))

# --------------------------------------------------
if __name__ == '__main__':
    main()
```

The template shows how to create named arguments for strings, integers, Booleans, as well as positional (unnamed) values. For this program, we want to know the min/max numbers to guess and the number of guesses allowed. We will provide reasonable defaults for all of them so that they will be completely optional. The thing I like best about this step is that it makes me think carefully about what I expect from the user. Change your program to this:

```
def get_args():
    """get args"""
    parser = argparse.ArgumentParser(description='Number guessing game')
    parser.add_argument('-m', '--min', help='Minimum value',
                        metavar='int', type=int, default=1)
    parser.add_argument('-x', '--max', help='Maximum value',
                        metavar='int', type=int, default=50)
    parser.add_argument('-g', '--guesses', help='Number of guesses',
                        metavar='int', type=int, default=5)
    return parser.parse_args()
```

Now in the `main` we will need to unpack the input:

```
def main():
    """main"""
    args = get_args()
    low = args.min
    high = args.max
    guesses_allowed = args.guesses
```

Alway assume you get garbage from the user, so let's check the input:

```
    if low < 1:
        print('--min cannot be lower than 1')
        sys.exit(1)

    if guesses_allowed < 1:
        print('--guesses cannot be lower than 1')
        sys.exit(1)

    if low > high:
        print('--min "{}" is higher than --max "{}"'.format(low, high))
        sys.exit(1)
```

The next thing we need is a random number between `--min` and `--max` for the user to guess. We can `import random` to do:

```
secret = random.randint(low, high)
```

The meat of the program will be an infinite loop where we keep asking the user:

```
prompt = 'Guess a number between {} and {} (q to quit): '.format(low, high)
```

Before we enter that loop, we'll need a variable to keep track of the number of guesses the user has made:

```
num_guesses = 0
```

The beginning of the play loop looks like this:

```
    while True:
        guess = input('[{}] {}'.format(num_guesses, prompt))
        num_guesses += 1
```

Here I want the user to know how many guesses they've made so far. We want to give them a way out, so they can enter "q" to quit:

```
        if guess == 'q':
            print('Now you will never know the answer.')
            sys.exit(0)
```

The input from the user will be a string, and we are going to need to convert it to an integer to see if it is the secret number. Before we do that, we must check that it is a digit:

```
        if not guess.isdigit():
            print('"{}" is not a number'.format(guess))
            continue
```

If it's not a digit, we `continue` to go to the next iteration of the loop. If we move ahead, then it's OK to convert the guess:

```
        print('You guessed "{}"'.format(guess))
        num = int(guess)
```

Now we need to determine if the user has guessed too many times, if the number if too high or low, or if they've won the game:

```
        if num_guesses >= guesses_allowed:
            print('Too many guesses! The number was "{}."'.format(secret))
            sys.exit()
        elif num < low or num > high:
            print('Number is not in the allowed range')
        elif num == secret:
            print('You win!')
            break
        elif num < secret:
            print('Too low.')
        else:
            print('Too high.')
```

The final version looks like this:

```
$ cat -n guess.py
     1	#!/usr/bin/env python3
     2	"""
     3	Author:  Ken Youens-Clark <kyclark@gmail.com>
     4	Purpose: Guess-the-number game
     5	"""
     6	
     7	import argparse
     8	import random
     9	import sys
    10	
    11	
    12	# --------------------------------------------------
    13	def get_args():
    14	    """get args"""
    15	    parser = argparse.ArgumentParser(
    16	        description='Guessing game',
    17	        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    18	
    19	    parser.add_argument(
        '-m',
        '--min',
        help='Minimum value',
        metavar='int',
        type=int,
        default=1)

    parser.add_argument(
        '-x',
        '--max',
        help='Maximum value',
        metavar='int',
        type=int,
        default=50)

    parser.add_argument(
        '-g',
        '--guesses',
        help='Number of guesses',
        metavar='int',
        type=int,
        default=5)

    return parser.parse_args()


# --------------------------------------------------
def main():
    """main"""
    args = get_args()
    low = args.min
    high = args.max
    guesses_allowed = args.guesses
    secret = random.randint(low, high)

    if low < 1:
        print('--min cannot be lower than 1')
        sys.exit(1)

    if guesses_allowed < 1:
        print('--guesses cannot be lower than 1')
        sys.exit(1)

    if low > high:
        print('--min "{}" is higher than --max "{}"'.format(low, high))
        sys.exit(1)
```

```python

    prompt = 'Guess a number between {} and {} (q to quit): '.format(low, high)
    num_guesses = 0

    while True:
        guess = input(prompt)
        num_guesses += 1

        if guess == 'q':
            print('Now you will never know the answer.')
            sys.exit(0)

        if not guess.isdigit():
            print('"{}" is not a number'.format(guess))
            continue

        print('You guessed "{}"'.format(guess))
        num = int(guess)

        if num_guesses >= guesses_allowed:
            print('Too many guesses! The number was "{}."'.format(secret))
            sys.exit()
        elif not low < num < high:
            print('Number is not in the allowed range')
        elif num == secret:
            print('You win!')
            break
        elif num < secret:
            print('Too low.')
        else:
            print('Too high.')


# --------------------------------------------------
if __name__ == '__main__':
    main()
```

Hangman

Here is an implementation of the game "Hangman" that uses dictionaries to maintain the "state" of the program – that is, all the information needed for each round of play such as the word being guessed, how many misses the user has made, which letters have been guessed, etc. The program uses the `argparse` module to gather options from the user while providing default values so that nothing needs to be provided. The `main` function is used just to gather the parameters and then run the `play` function which recursively calls itself, each time passing in the new "state" of the program. Inside `play`, we use the `get` method of `dict` to safely ask for keys that may not exist and use defaults. When the user finishes or quits, `play` will simply call `sys.exit` to stop. Here is the code:

```
$ cat -n hangman.py
     1	#!/usr/bin/env python3
     2	"""
     3	Author:  Ken Youens-Clark <kyclark@gmail.com>
     4	Purpose: Hangman game
     5	"""
     6	
     7	import argparse
     8	import os
     9	import random
    10	import re
    11	import sys
    12	
    13	
    14	# --------------------------------------------------
    15	def get_args():
    16	    """parse arguments"""
    17	    parser = argparse.ArgumentParser(
    18	        description='Hangman',
    19	        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    20	
    21	    parser.add_argument(
    22	        '-l', '--maxlen', help='Max word length', type=int, default=10)
    23	
    24	    parser.add_argument(
    25	        '-n', '--minlen', help='Min word length', type=int, default=5)
    26	
    27	    parser.add_argument(
    28	        '-m', '--misses', help='Max number of misses', type=int, default=10)
    29	
    30	    parser.add_argument(
    31	        '-w',
    32	        '--wordlist',
    33	        help='Word list',
    34	        type=str,
    35	        default='/usr/share/dict/words')
    36	
    37	    return parser.parse_args()
    38	
```

```

# --------------------------------------------------
def main():
    """main"""
    args = get_args()
    max_len = args.maxlen
    min_len = args.minlen
    max_misses = args.misses
    wordlist = args.wordlist

    if not os.path.isfile(wordlist):
        print('--wordlist "{}" is not a file.'.format(wordlist))
        sys.exit(1)

    if min_len < 1:
        print('--minlen must be positive')
        sys.exit(1)

    if not 3 <= max_len <= 20:
        print('--maxlen should be between 3 and 20')
        sys.exit(1)

    if min_len > max_len:
        print('--minlen ({}) is greater than --maxlen ({})'.format(
            min_len, max_len))
        sys.exit(1)

    regex = re.compile('^[a-z]{' + str(min_len) + ',' + str(max_len) + '}$')
    words = [w for w in open(wordlist).read().split() if regex.match(w)]
    word = random.choice(words)
    play({'word': word, 'max_misses': max_misses})


# --------------------------------------------------
def play(state):
    """Loop to play the game"""
    word = state.get('word') or ''

    if not word:
        print('No word!')
        sys.exit(1)

    guessed = state.get('guessed') or list('_' * len(word))
    prev_guesses = state.get('prev_guesses') or set()
    num_misses = state.get('num_misses') or 0
    max_misses = state.get('max_misses') or 0
    if ''.join(guessed) == word:
        msg = 'You win. You guessed "{}" with "{}" miss{}!'
        print(msg.format(word, num_misses, '' if num_misses == 1 else 'es'))
        sys.exit(0)

    if num_misses >= max_misses:
        print('You lose, loser!  The word was "{}."'.format(word))
        sys.exit(0)

    print('{} (Misses: {})'.format(' '.join(guessed), num_misses))
    new_guess = input('Your guess? ("?" for hint, "!" to quit) ').lower()

    if new_guess == '!':
        print('Better luck next time, loser.')
        sys.exit(0)
    elif new_guess == '?':
        new_guess = random.choice([x for x in word if x not in guessed])
        num_misses += 1

    if not re.match('^[a-zA-Z]$', new_guess):
        print('"{}" is not a letter'.format(new_guess))
        num_misses += 1
    elif new_guess in prev_guesses:
        print('You already guessed that')
    elif new_guess in word:
        prev_guesses.add(new_guess)
        last_pos = 0
        while True:
            pos = word.find(new_guess, last_pos)
            if pos < 0:
                break
            elif pos >= 0:
                guessed[pos] = new_guess
                last_pos = pos + 1
    else:
        num_misses += 1

    play({
        'word': word,
        'guessed': guessed,
        'num_misses': num_misses,
        'prev_guesses': prev_guesses,
        'max_misses': max_misses
    })

```

```

# ---------------------------------------------------
if __name__ == '__main__':
    main()
```