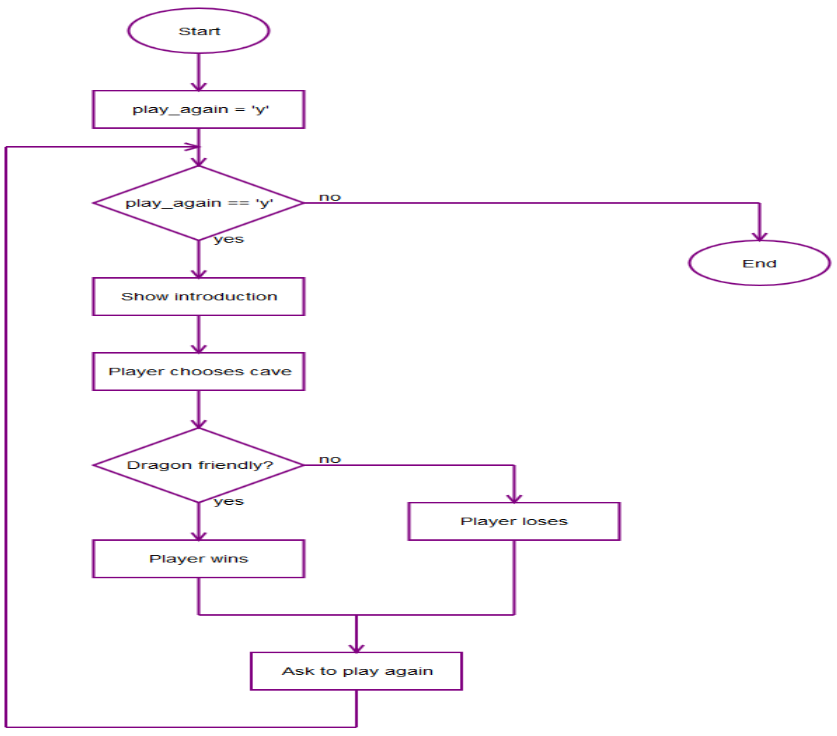
Tutorial: Functions (Dragon Kingdom Game)

The aim of this tutorial is to create a simple game that demonstrates the use of user-defined functions (with arguments, parameters, return values and local variables). We will also review flowcharts, the Boolean *and* operator and while loops.

Dragon Kingdom Game - The player is in the Kingdom of Dragons. The dragons live in caves with their large piles of collected treasure. Some dragons are friendly and share their treasure. Other dragons are hungry and eat anyone who enters their cave. The player approaches two caves, one with a friendly dragon and the other with a hungry dragon, but doesn't know which dragon is in which cave and must choose between the two caves.

View of game when it is run. The player's input is in bold. The flowchart is below.

```
You are in the Kingdom of Dragons. In front of you,
you see two caves. In one cave, the dragon is friendly
and will share his treasure with you. The other dragon
is hungry and will eat you on sight.
Which cave will you go into? (1 or 2)
1
You approach the cave...
A large dragon jumps out in front of you!
He opens his jaws and...
Gobbles you down!
Do you want to play again? (y or n)
n
```



Change the wording of the game if you wish but keep the flowchart structure shown above.

1. Create a new program called *dragon.py*.
2. Your program will need to import two modules, so place these at the top of the program. Random provides *randint()* and time provides the sleep() function that we will use.

```
import random
import time
```

3. **Show Introduction.** Define a function called *display_intro* that has no parameters and prints out the following when called.

```
You are in the Kingdom of Dragons.  In front of you,
you see two caves. In one cave, the dragon is friendly
and will share his treasure with you. The other dragon
is hungry and will eat you on sight.
```

- Note: Use three quotes around a string for *multiline strings*.
- Remember, a function's definition must come *before* you call the function.
- Add a call to the function in your program to check that it works as expected.

4. **Player Choses Cave**. Define a function called *choose_cave* that has no parameters and contains the following code. This function will ask the player if they want to enter cave 1 or 2 and returns a valid response.

```
cave = ''     # local variable with empty string
while cave != '1' and cave != '2':
     print('Which cave will you go into? (1 or 2)')
     cave = input()
return cave
```

- The variable *cave* is created inside the function's **local scope**. It will be forgotten when *choose_cave()* returns and re-created if *choose_cave()* is called again.
- The while condition carries out *input validation* by looping until 1 or 2 is entered.
- The condition contains the Boolean *and* operator.
  - The condition is only true if both parts are True.
  - If either or both of the Boolean values are False, then the condition is False.
- If this is not clear, try entering the following expressions into the interactive shell:

```
>>> True and True    # invalid input
True
>>> True and False   # 2 entered
False
>>> False and True   # 1 entered
False
```

- What happens if you change the code to the use integers as follows:

```
while cave != 1 and cave != 2:
```

  - Why are the condition values '1' or '2' shown as strings in the original code?

- In the main part of the program create a variable called *cave_number* to save the value returned by the *choose_cave()* function.

```
cave_number  = choose_cave()
```

- You can add a temporary print() to check that only a 1 or 2 is returned when testing.

```
print(cave_number )   # check 1 or 2 returned
```

5. **Check for Friendly Dragon**
   - Now create a function called *check_cave* that has a parameter called *chosen_cave*
   - Add the following code to the *check_cave* function:

```
print('You approach the cave...')
time.sleep(2)
print('A large dragon jumps out in front of you! ')
print('He opens his jaws and...')
time.sleep(2)
```

   - The time module function sleep() is used to pause the program for 2 seconds (to build some suspense in the game).
   - Add the following code to the *check-cave* function. The randint() function will return either 1 or 2 and store this in friendlyCave to indicate the cave with the friendly dragon.

```
friendlyCave = random.randint(1, 2)
```

   - Create a condition to check if the player wins or loses.
     - o The player's chosen cave (stored in chosen_cave) is equal to the randomly generated value (stored in friendlyCave). Print 'Gives you his treasure!'
     - o Otherwise, print 'Gobbles you down!'
     - The value in friendlyCave is an integer because randint() returns integers. The value in chosen_cave is a string. Convert friendlyCave to compare two strings.
     - Then, add a call to the check_cave function passing the cave_number as an argument.

6. **Calling the Functions.** The main part of your program should now look like this.

```
display_intro()
cave_number = choose_cave()
check_cave(cave_number)
```

7. **Ask to Play Again.** Add the feature to check if the user wants to play again (see flowchart).

8. **FINAL CHALLENGE -** Extend the above program. Create another program *dragon_multi.py* that has 4 caves (instead of 2) and include appropriate messages for the player selecting cave 1, 2, 3 or 4. Draw the amended flowchart first, then create the program.