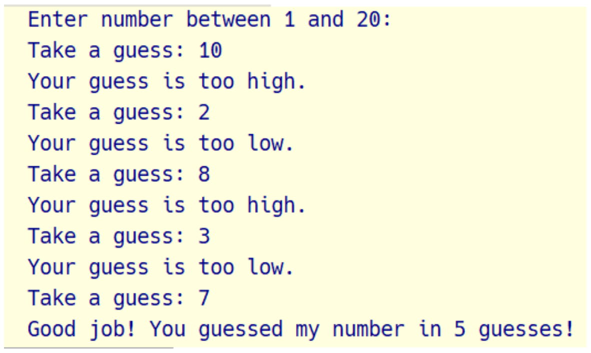
# Short program: Guess the Number

- We're going to bring the last few topics together in a complete little game script. The same mechanics will be required for the "Rock, Paper, Scissors" home programming assignment
- This example also demonstrates an exemplary solution path:
    1. Understand what's asked from you (requirements)
    2. Understand what the program needs from you (input)
    3. Understand what's the result supposed to look like (output)
    4. Write the process as pseudocode (without syntax)
    5. Create a process diagram (with commands)
    6. Code the Python program (source code)
    7. Run, test and debug the source code
    8. Fix pseudocode/diagram accordingly.
    9. Identify extensions.
    10. Implement extensions (repeat steps 4-8).

- Write a 'Guess the number' game. When you run the program, the output should look like this:



```
Enter number between 1 and 20:
Take a guess: 10
Your guess is too high.
Take a guess: 2
Your guess is too low.
Take a guess: 8
Your guess is too high.
Take a guess: 3
Your guess is too low.
Take a guess: 7
Good job! You guessed my number in 5 guesses!
```

Figure 1: Desired output of guessTheNumber.py

- The program should generate a random number between 1 and 20.
- Enter the source code into the IDLE file editor, or into Colab, and save as `guessTheNumber.py`.
- Solution path/pseudocode (code highlighted)
    1. `import` `random` module.
    2. Generate a `random` number.
    3. Store number in `num`.
    4. Set `attempt` (number of guesses) to 0.
    5. Get `input` number `guess` from user.
    6. Increase `attempt` by 1
    7. Check if `guess` is the same as `num`
    8. Print success message and `attempt` value
    9. End program
    10. Otherwise, check if `guess` is smaller than `num`
    11. Print information
    12. Otherwise, check if `guess` is larger than `num`
    13. Print information
    14. Return to step 3

- BPMN Process diagram:

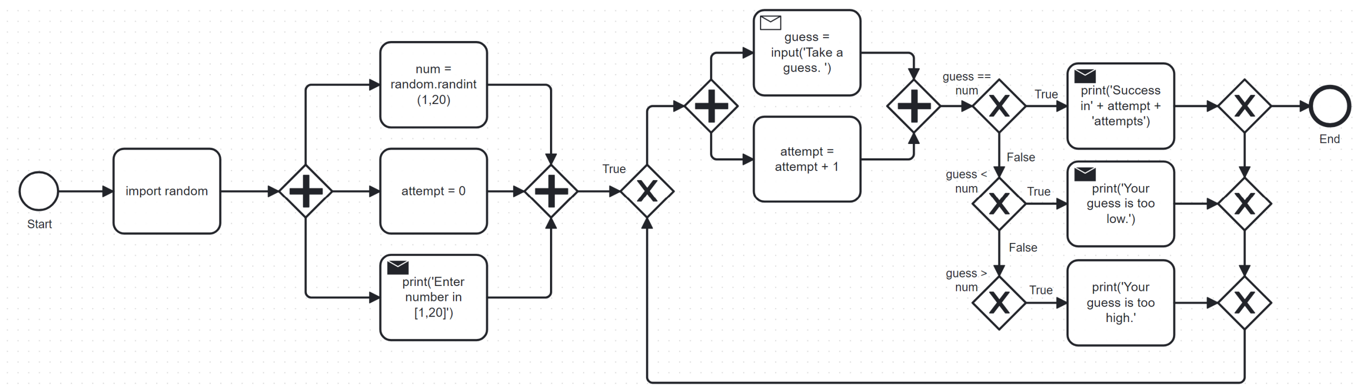

Start
import random
num = random.randint (1,20)
attempt = 0
print('Enter number in [1,20]')
True
guess = input('Take a guess. ')
attempt = attempt + 1
guess == num
True
print('Success in' + attempt + 'attempts')
End
False
guess < num
True
print('Your guess is too low.')
False
guess > num
True
print('Your guess is too high.'

Figure 2: Flow diagram for guessTheNumber.py

- Solution Python code (16 + 5 lines):

```python
# import random module
import random
# pick random number between 1 and 20
num = random.randint(1,20)
# set attempts counter to 0
attempt = 0
# ask user for number guess
print('Enter number between 1 and 20: ')
# infinite loop until number is guessed
while True:
    guess = int(input('Take a guess: '))
    attempt = attempt + 1
    if guess < num:
        print('Your guess is too low.')
        continue
    elif guess > num:
        print('Your guess is too high.')
        continue
    else:
        print('Good job! You guessed my number in ' + str(attempt) + ' guesses!')
        break
```

- Program extensions:
    1. Make program safe against no/wrong input (exception handling): currently, it terminates with an error if a floating-point number or a letter or nothing is entered by mistake.
    2. Exchange the infinite `while` loop by a `for` loop with a set number of allowed guesses (most games don't go on forever).
- What did you learn?
    1. For best productivity and learning, follow a solution path - don't just "code away"
    2. For best learning effects find different solutions to the same problem.
    3. For best results, handle exceptions. Balance exception handling with usability and performance.
    4. There is always more than one solution, usually many. There is no best solution to a programming problem that satisfies all requirements, even the unspoken ones, equally well.

Created: 2023-10-30 Mon 10:22