pragmatic_introduction_to_python_language_2

February 21, 2021

1 Coding Practice for Beginners

1.1 What's the target

With the following exercises, you will learn to:

- declare variables
- collect user input
- store information
- repeat an action through loops
- write functions to repeat blocks of code

1.2 How to use these projects

First, read through the instructions and make sure you understand what you've read. Try to say what you read in your own words. Attempt to solve it on your own without going through YouTube tutorials or Google search. Push yourself. This is the idea of **deliberate practice** from Behavioral Psychology.

If you haven't made any progress at all, watch a YouTube tutorial where available then look through the example code. You can also search online to see more example tutorials and Python code for the same problem. After going through, go back and try to write the code by yourself without looking at the tutorial. Again, push yourself.

Whatever you do, do not blindly copy out the code in the tutorial. You may finish quickly but the reality is you haven't learned anything. If you feel you're completely stuck after lots of attempts, take a break. When you step away, we know from Neuroscience research that your subconscious will continue the learning. This is because your mind has shifted from **focused mode** to **diffused mode**.

Finally pat yourself on the back (honestly I don't known if this is a correct english phrase)

Repeat, repeat, repeat… and if you want to know more about *Deliberate Practice* click here. You can find the source of many, if not all, of this projects here.

Question: Why does the blog author describe exaclty **42** projects?

1.3 Word count

Ask the user what's on their mind. Then after the user responds, count the number of words in the sentence and print that as an output.

Example:

Prompt: what's on your mind today? Input: well, it's just a day for me to be an expert in coding Output: oh nice, you just told me what's on your mind in 13 words!

```
[ ]: answer = input("What's in your mind today?    ")
```

```
[ ]: print(("oh nice, you just told me what's on your mind in %s words!") %␣
     ↪str(len(answer)))
```

To take this a step further, open a file that is handed to you, count the number of words in there, then print it out.

1.4 What's my acronym?

Ask the user to enter the full meaning of an organization or concept and you'll provide the acronym to the user.

For example:

- Input -> As Soon As Possible. Output -> ASAP.
- Input -> World Health Organization. Output -> WHO.
- Input -> Absent Without Leave. Output -> AWOL.

```
[ ]: acronim = ''

x = input("Insert your sentence : ")
y = x.split()

for s in y:
    acronim = acronim + s[0].upper()
```

```
[ ]: print(acronim)
```

1.5 Leap Year

1.5.1 A little digression on time measure

The date system currently in use is based on the Gregorian calendar, first established in 1582 by Pope Gregory XIII. This calendar was designed to correct the errors introduced by the less accurate Julian calendar. In the Gregorian calendar, a normal year consists of 365 days. Because the actual length of a sidereal year (the time required for the Earth to revolve once about the Sun) is actually 365.2425 days, a "leap year" of 366 days is used once every four years to eliminate the error caused by three normal (but short) years. Any year that is evenly divisible by 4 is a leap year: for example, 1988, 1992, and 1996 are leap years.

However, there is still a small error that must be accounted for. To eliminate this error, the Gregorian calendar stipulates that a year that is evenly divisible by 100 (for example, 1900) is a leap year only if it is also evenly divisible by 400.

For this reason, the following years **are not** leap years:

1700, 1800, 1900, 2100, 2200, 2300, 2500, 2600

This is because **they are evenly divisible by 100 but not by 400**.

The following years **are** leap years: *1600, 2000, 2400*

This is because **they are evenly divisible by both 100 and 400**.

Ok, now let's do a simple python program to find leap years.

To understand this example, you should have the knowledge of the following Python programming topics:

- Python Operators
- Python if...else Statement

[ ]:
```
# Python program to check if year is a leap year or not

# To get year (integer input) from the user
year = int(input("Enter a year: "))

if (year % 4) == 0:
   if (year % 100) == 0:
       if (year % 400) == 0:
           print("{0} is a leap year".format(year))
       else:
           print("{0} is not a leap year".format(year))
   else:
       print("{0} is a leap year".format(year))
else:
   print("{0} is not a leap year".format(year))
```

1.6 Is it a Fibonacci number?

You input a number and the function created checks whether the number belongs to the Fibonacci sequence or not. If you don't known what a Fibonacci number is and how to detect it, read this page to find out some suggestion.

[ ]:
```
import math

# A utility function that returns true if x is perfect square
def isPerfectSquare(x):
    s = int(math.sqrt(x))
    return s*s == x

# Returns true if n is a Fibinacci Number, else false
def isFibonacci(n):
    # n is Fibinacci if one of 5*n*n + 4 or 5*n*n - 4 or both
    # is a perferct square
    return isPerfectSquare(5*n*n + 4) or isPerfectSquare(5*n*n - 4)
```

```
# A utility function to test above functions
for i in range(1,15):
    print(i, isFibonacci(i))
```

1.7 Guess the number

You ask a user to guess a number between 1 and 50. If they guess outside that range, you prompt with an error encouraging them to choose a number within the proper range. Whenever they guess the wrong number you ask if they want to keep playing or if they'd like to quit.

Finally, when the user eventually guesses the right number you congratulate them and show the number of attempts they had.

random module

Sometimes we want the computer to pick a random number in a given range, pick a random element from a list, pick a random card >from a deck, flip a coin, etc. The random module provides access to functions that support these types of operations. One such >operation is random.randrange(a, b) method (returns a random integer between a and b) that we are going to use in order to >select one random number.

[ ]:
```
import random

number = random.randrange(1, 100)
guess  = input("Guess a number between 1 and 100 : ")
guess  = int(guess)

number_of_guess = 0
while guess != number:
    if(guess < number):
        print("You need to guess higher. Try again")
        guess  = int(input("Guess a number between 1 and 100 : "))
    if(guess > number):
        print("You need to guess lower. Try again")
        guess  = int(input("Guess a number between 1 and 100 : "))
    number_of_guess = number_of_guess + 1

print("You find the correct number with " + str(number_of_guess) + " guesses")
```

1.8 Guess the word

1.8.1 First Game

In this game, there is a list of words present, out of which python interpreter will choose 1 random word. The user first has to input their names and then, will be asked to guess any characters. If the random word contains that character, it will be shown as the output(with correct placement) else the program will ask you to guess another character. User will be given 12 turns(can be changed accordingly) to guess the complete word.

To understand this example, you should have the knowledge of the following Python programming topics:

- using random numbers
- using strings
- loops and conditional(If, else) statements.

[ ]:
```
import random
# library that we use in order to choose
# on random words from a list of words

name = input("What is your name? ")
# Here the user is asked to enter the name first

print("Good Luck ! ", name)

words = ['rainbow', 'computer', 'science', 'programming', 'python',␣
 ↪'mathematics', 'player', 'condition', \
         'reverse', 'water', 'board', 'geeks']

# Function will choose one random
# word from this list of words
word = random.choice(words)


print("Guess the characters")

guesses = ''

# any number of turns can be used here
turns = 12

while turns > 0:
    # counts the number of times a user fails
    failed = 0

    # all characters from the input
    # word taking one at a time.
    for char in word:

        # comparing that character with
        # the character in guesses
        if char in guesses:
            print(char)

        else:
            print("_")
```

```
            # for every failure 1 will be
            # incremented in failure
            failed += 1

    if failed == 0:
        # user will win the game if failure is 0
        # and 'You Win' will be given as output
        print("You Win")

        # this print the correct word
        print("The word is: ", word)
        break

    # if user has input the wrong alphabet then
    # it will ask user to enter another alphabet
    guess = input("guess a character:")

    # every input character will be stored in guesses
    guesses += guess

    # check input with the character in word
    if guess not in word:

        turns -= 1

        # if the character doesn't match the word
        # then "Wrong" will be given as output
        print("Wrong")

        # this will print the number of
        # turns left for the user
        print("You have", + turns, 'more guesses')


        if turns == 0:
            print("You Loose")
```

1.8.2 Second Game

In the next we still focus on the user having to guess a randomly choosen word but the game is different from the previous one. You can create a list from which the word would have to be guessed. I propose to save the list in a txt file that you can load at the beginning of the game. In the example below I create two differente files with the 1000 most common used words in english and italian.

The game starts with only the first character of the randomly chosen word and a list of ' * ' for the remaining characters. After this, every time the user have a guess, another character of the secret word, at random, is shown until the complete word is revealed.

You can improve the game setting a cap on the number of guesses allowed.

To understand this example, you should have the knowledge of the following Python programming topics:

- using random numbers
- reading txt files
- loops and conditional(If, else) statements

Click here for the documentation of the widgets library.

[ ]:
```
import ipywidgets as widgets

language = widgets.Select(
    options=['english', 'italian'],
    value='english',
    # rows=10,
    description='language :',
    disabled=False
)
display(language)
```

[ ]:
```
print(language.value)
```

[ ]:
```
# define path
path       = './data/txt/'
# takes ongly the first two letters of the chosen language
suffix     = language.value[:2]
# build complete file name
file_name  = path + '1000_most_common_words_' + suffix + '.txt'

with open(file_name, "r") as f:
    words_list = f.readlines()

# chose a random word from the list
from random import randint

item        = randint(0, len(words_list))
secret_word = words_list[item].strip()
start       = secret_word[0] + ('*' * (len(secret_word)-1))
guess       = ''

extracted   = []
for l in range(1, len(secret_word)):
    print(start)
    temp      = list(start)
    guess     = input('Your guess? ')
    # if the guess is correct we print a message to the winner and
    # stop the for loop with the instruction 'break'
```

```
    if guess == secret_word:
        print("You won! The secret word was : " + secret_word)
        break
    # select a random integer between 1 and the secret word lenght
    k         = randint(1, len(secret_word)-1)
    # we check if the random number has been already drawn, in this
    # case it is saved in the extracted list, so we check if k is
    # in extracted in that case we draw another number.
    while k in extracted and l < len(secret_word) - 1:
        k  = randint(1, len(secret_word)-1)

    temp[k]    = secret_word[k]
    start      = ''.join(temp)
    extracted.append(k)

if l == len(secret_word)-1:
    print("You lost! The secret word was : " + secret_word)
```

1.9 Rock, Paper, Scissors

This is a popular game played between two people. Each player gets to form one of three shapes using their hand:

- rock
- paper
- scissors

below you can find a simple program that implements this game in python. However there is no check on input, please correct this gap!

[ ]:
```
from random import randint

choice = ["rock","paper","scissors"]
player = ''

while player != 'end':
    computer = choice[randint(0,2)]

    player = input("Your Choice: ").lower()
    if(player != 'end'):
       print("Computer Chose: " + computer)

    if player == computer:
        print("Draw")
    elif player == "rock" and computer == "paper":
        print("Computer Wins")
    elif player == "rock" and computer == "scissors":
        print("Player Wins")
```

```
    elif player == "paper" and computer == "rock":
        print("Player Wins")
    elif player == "paper" and computer == "scissors":
        print("Computer Wins")
    elif player == "scissors" and computer  == "rock":
        print("Computer Wins")
    elif player == "scissors" and computer == "paper":
        print("Player Wins")
    elif player == "end":
        print("Thank you for playing with this notebook!")
```

1.10 Mad Libs Game

The idea is very simple: ask the user for an input, this could be anything such as a name, an adjective, a pronoun or even an action. Once you get the input, you can rearrange it to build up your own story. **Primarily focused on strings, variables, and concatenation, this project will teach you how to manipulate user-inputted data**. Of course you need a template to start with, for example, in the wikipedia mad libs page you can find this one:

[ ]:
```
exclamation = input('Insert an exclamation      :')
adverb      = input('Give me an adverb...       :')
noun        = input('... and a noun, please     :')
adjective   = input('and finally and adjective  :')

mad = '%s! he said %s as he jumped into his convertible %s and drove off with␣
 ↪his %s wife'

mad = mad % (exclamation, adverb, noun, adjective)
```

[ ]:
```
print(mad)
```

Here you can find much more examples. You can download a pdf with the template of a Mad Lib and try to complete, for example ...

[ ]:
```
adj1 = 'adorable'
adj2 = 'adventurous'
adj3 = 'aggressive'
noun1 = 'cat'
noun2 = 'dog'
plural1 = 'cars'
plural2 = 'windows'
plural3 = 'trains'
plural4 = 'trees'
game1 = 'dungeons and dragons'
game2 = 'monopoli'
ing_verb_1 = 'eating'
ing_verb_2 = 'phishing'
```

```
ing_verb_3 = 'sending'
number     = '42'
```

[ ]:
```
template = 'A vacation is when you take a trip to some ' + adj1 + ' place '
 ↪        + \
           'with your ' + adj2 + ' family. Usually you go to some place '
 ↪        + \
           'that is near a/an ' + noun1 + ' or up on a/an ' + noun2 + '.'
 ↪        + \
           'A good vacation place is one where you can ride ' + plural1 + ' '
 ↪        + \
           'or play ' + game1 + ' or go hunting for ' + plural2 + '. I like '
 ↪        + \
           'to spend my time ' + ing_verb_1 + ' or ' + ing_verb_2 + '. '
 ↪        + \
           'When parents go on a vacation, they spend their time eating '
 ↪        + \
           'three ' + plural3 + ' a day and fathers play ' + game2 + ', and
↪mothers '   + \
           'sit around ' + ing_verb_3 + '. Parents '
 ↪        + \
           'need vacations more than kids because parents are always very '
 ↪        + \
           '' + adj3 + ' and because they have to work ' + number + ' '
 ↪        + \
           'hours every day all year making enough ' + plural4 + ' to pay '
 ↪        + \
           'for the vacation.'

print(template)
```

1.11 Biography info

Ask a user for their personal information one question at a time. Then check that the information they entered is valid. Finally, print a summary of all the information they entered back to them.

> *Example*: What is your name? If the user enters * you prompt them that the input is wrong, and ask them to enter a valid name.

It mandatory that you check the validity of the input data! At the end, if the input data are ok, you should print a short summary, otherwise you should print an error message with a guess on the probable error.

[ ]:
```
#
# phase 1 collect infos
#
name     = input("What's your name?                                    : ")
```

```
surname  = input("What's your surname?                                          : ")
birth    = input("When were you born (enter date in format yyyy/mm/dd)?     : ")
email    = input("Your email?                                                  : ")
```

1.11.1 Check a date format

First of all, we check if the date is valid or not. A date is called valid if it actually exists in the calendar. For checking the validity of the date, we will use one python module called 'datetime'. This module doesn't provide any dedicated method to check if a date is valid or not but we will use this module with a simple trick to find out if a date is valid or not. Before going into details, let me quickly introduce you to the datetime module.

Python datetime module Python datetime module is one of the most useful modules to work with simple and complex time. We can import this module to a python program by using the 'import datetime' statement at the beginning of the program. This module provides a lot of different methods to work with time. For example, we can use this module to print the current time, add days to the current time, add hours to the current time, add minutes to the current time etc. 'datetime' module can work with 'naive' and 'aware' kind of date time objects. 'aware' objects can hold additional information with the date time value like daylight saving information etc. As I have explained above, it doesn't provide any method to check the validity of a date. We will use its constructor to create one 'datetime' object using the user provided values. If the constructor fails, it will throw one error. We will assume that the input values are not valid in that case.

The The Algorithm to use

- Get the input from the user
- Input should be in the form of yyyy/mm/dd
- Extract the inputs in different variables. e.g. if the user input is 02/04/99, we will extract the numbers 02, 04, and 99 and store them in three different variables.
- Use the constructor of 'datetime' module to check if the date is valid or not.

[ ]:
```
import datetime

inputDate = birth

year, month, day = inputDate.split('/')

isValidDate = True
try :
    datetime.datetime(int(year),int(month),int(day))
except ValueError :
    isValidDate = False

if(isValidDate) :
    print ("Input date is valid ..")
else :
    print ("Input date is not valid..")
```

Let's transform this code into a function...

[ ]:
```
def check_date(inputDate):
    year, month, day = inputDate.split('/')

    isValidDate = True
    try :
        datetime.datetime(int(year),int(month),int(day))
    except ValueError :
        isValidDate = False

    return isValidDate
```

[ ]:
```
#
# phase 2 check infos
#
correct     = True
wrong_input = ''
# check name
name = name + ' ' + surname
if(name.find('*') > 0 or len(name.strip()) == 0):
    correct = False
    wrong_input = 'name'

# check birth date
if(not check_date(birth)):
    correct = False
    wrong_input = 'birth date'

# check email
# An email is a string (a subset of ASCII characters) separated into two parts
# by @ symbol, a "personal_info" and a domain, that is personal_info@domain.
if(email.find("@") > 0):
    items = email.split('@')
    if(items[1].find(".") < 0):
        correct = False
        wrong_input = 'e-mail address'
else:
    correct = False
    wrong_input = 'e-mail address'
```

[ ]:
```
#
# phase 3 print final results
#
if(correct):
    print(("Name          : %s %s") % (name, surname))
    print(("Date of birth : %s")    % birth)
    print(("Your e-mail   : %s")    % email)
```

```
else:
    print('Error in input data. Please check your ' + wrong_input)
```

Improve the validation phase checking if the input year is a leap year.

1.12 Is it a palindrome?

Ask the user to give you a few words. Then check if any of the words is a palindrome. A palindrome is a word that remains the same whether it's read forward or backward.

[ ]:
```
# Python sequence slice addresses can be written as a[start:end:step]
original_list = list(range(1,22))
original_list
```

[ ]:
```
print(original_list[2:15:3])
```

[ ]:
```
# When start and end are absent, the whole sequence is assumed and thus s[:k]␣
 ↪means "every k-th item
print(original_list[::4])
```

[ ]:
```
# you can use negative step to reverse a list...
print(original_list[::-1])
```

[ ]:
```
word = input("Enter a word : ")
```

[ ]:
```
reversed = word[::-1]

if(word == reversed):
    print("The word you have entered is a palindrome!")
else:
    print("The word you have entered is not a palindrome!")
```

[ ]:
```
text = "sator arepo tenet opera rotas"

original_list = text.split()
reversed_list = original_list[::-1]
reversed_list
```

[ ]:
```
check_palindrome = []
for word in reversed_list:
    check_palindrome.append(word[::-1])
print(check_palindrome)

if(check_palindrome == original_list):
    print("\nThe text you have entered is a palindrome!")
```

1.13 Conclusion and References

- The documentation for the ipywidgets library is here
- *42 Exciting Python Project Ideas & Topics for Beginners* by Rohit Sharma link