REPUBLIC OF THE PHILIPPINES
POLYTECHNIC UNIVERSITY OF THE PHILIPPINES
STA. MESA, MANILA

COLLEGE OF ENGINEERING

ELECTRONICS ENGINEERING DEPARTMENT

CMPE 20022: COMPUTER PROGRAMMING

INSTRUCTIONAL MATERIAL

MODULE 3 – LAB ACTIVITY

Module 3

Laboratory Activity # 1

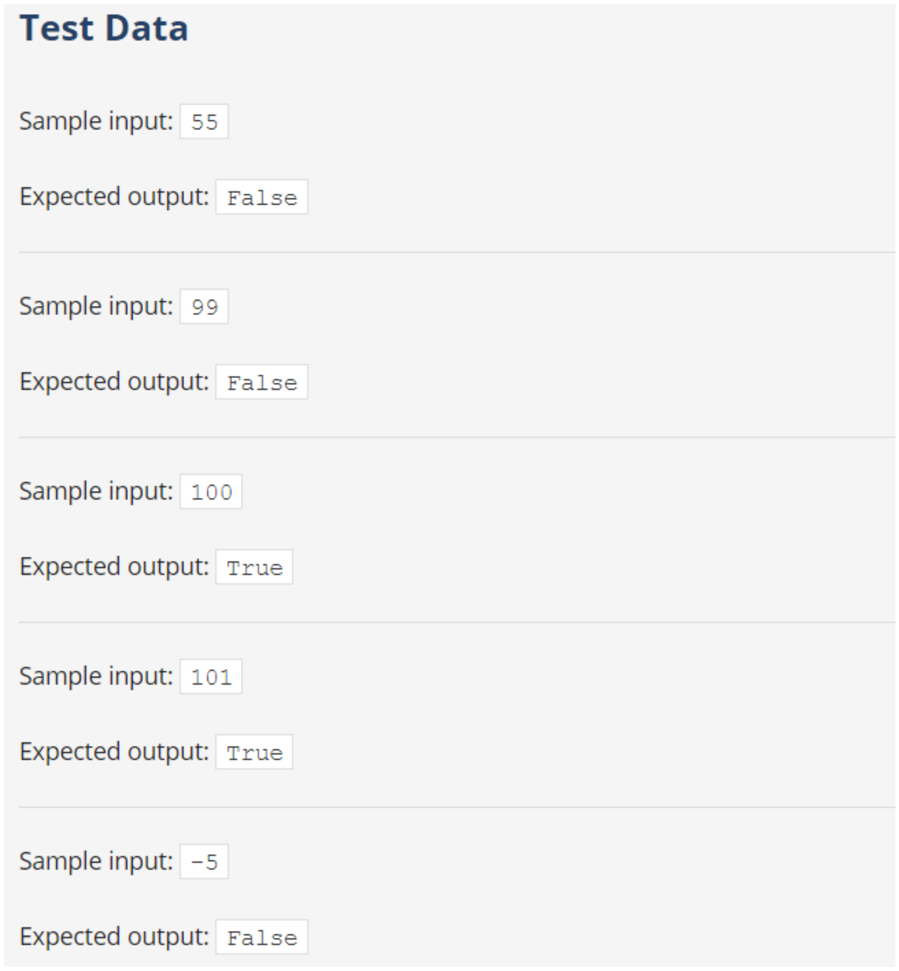
Objectives

- becoming familiar with the the `input()` function;
- becoming familiar with comparison operators in Python.

Scenario

Using one of the comparison operators in Python, write a simple two-line program that takes the parameter `n` as input, which is an integer, and prints `False` if `n` is less than `100`, and `True` if `n` is greater than or equal to `100`.

Don't create any `if` blocks (we're going to talk about them very soon). Test your code using the data we've provided for you.



Test Data

Sample input: `55`

Expected output: `False`

Sample input: `99`

Expected output: `False`

Sample input: `100`

Expected output: `True`

Sample input: `101`

Expected output: `True`

Sample input: `-5`

Expected output: `False`

Laboratory Activity # 2

Objectives

- becoming familiar with the the `input()` function;
- becoming familiar with comparison operators in Python;
- becoming familiar with the concept of conditional execution.

Scenario

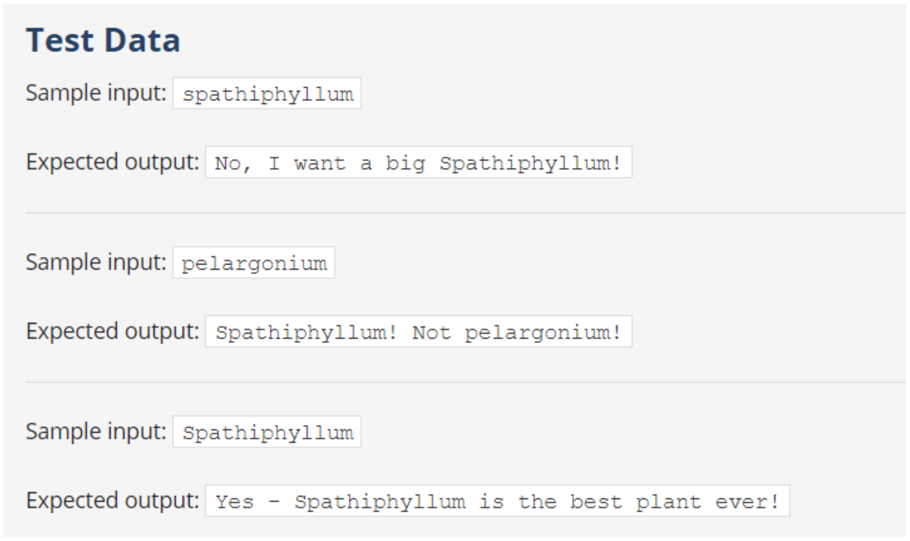
Spathiphyllum, more commonly known as a peace lily or white sail plant, is one of the most popular indoor houseplants that filters out harmful toxins from the air. Some of the toxins that it neutralizes include benzene, formaldehyde, and ammonia.

Imagine that your computer program loves these plants. Whenever it receives an input in the form of the word `Spathiphyllum`, it involuntarily shouts to the console the following string: `"Spathiphyllum is the best plant ever!"`

Write a program that utilizes the concept of conditional execution, takes a string as input, and:

- prints the sentence `"Yes - Spathiphyllum is the best plant ever!"` to the screen if the inputted string is `"Spathiphyllum"` (upper-case)
- prints `"No, I want a big Spathiphyllum!"` if the inputted string is `"spathiphyllum"` (lower-case)
- prints `"Spathiphyllum! Not [input]!"` otherwise. Note: `[input]` is the string taken as input.

Test your code using the data we've provided for you. And get yourself a Spathiphyllum, too!



Test Data

Sample input: `spathiphyllum`

Expected output: `No, I want a big Spathiphyllum!`

Sample input: `pelargonium`

Expected output: `Spathiphyllum! Not pelargonium!`

Sample input: `Spathiphyllum`

Expected output: `Yes - Spathiphyllum is the best plant ever!`

Laboratory Activity # 3

Objectives

Familiarize the student with:

- using the *if-else* instruction to branch the control path;
- building a complete program that solves simple real-life problems.

Scenario

Once upon a time there was a land - a land of milk and honey, inhabited by happy and prosperous people. The people paid taxes, of course - their happiness had limits. The most important tax, called the *Personal Income Tax* (*PIT* for short) had to be paid once a year, and was evaluated using the following rule:

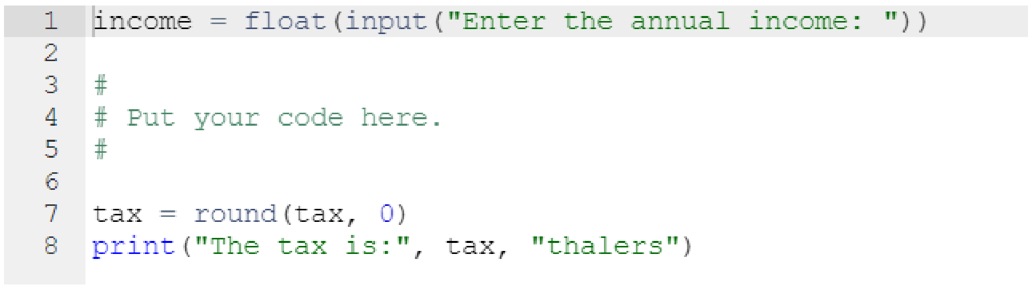
- if the citizen's income was not higher than 85,528 thalers, the tax was equal to 18% of the income minus 556 thalers and 2 cents (this was the so-called *tax relief*)
- if the income was higher than this amount, the tax was equal to 14,839 thalers and 2 cents, plus 32% of the surplus over 85,528 thalers.

Your task is to write a **tax calculator**.

- It should accept one floating-point value: the income.
- Next, it should print the calculated tax, rounded to full thalers. There's a function named `round()` which will do the rounding for you - you'll find it in the skeleton code in the editor.

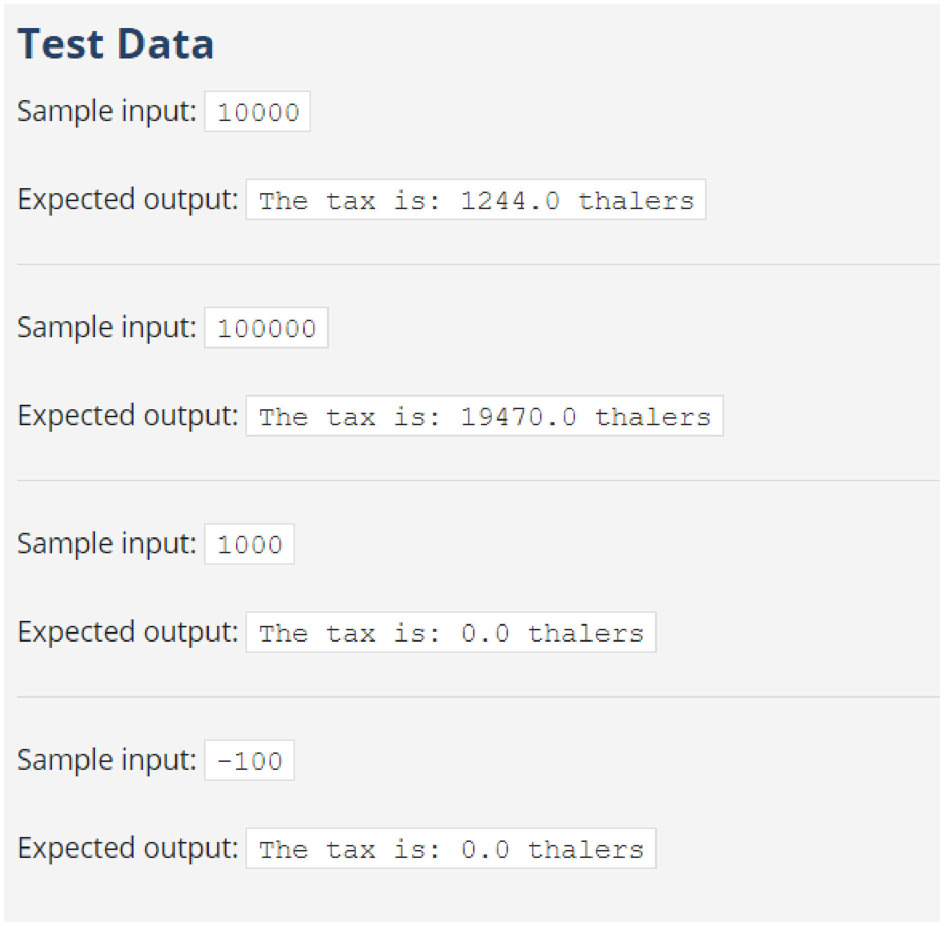
Note: this happy country never returns money to its citizens. If the calculated tax is less than zero, it only means no tax at all (the tax is equal to zero). Take this into consideration during your calculations.

Look at the code in the editor - it only reads one input value and outputs a result, so you need to complete it with some smart calculations.



```
income = float(input("Enter the annual income: "))

#
# Put your code here.
#

tax = round(tax, 0)
print("The tax is:", tax, "thalers")
```

Test your code using the data we've provided.



Test Data

Sample input: `10000`

Expected output: `The tax is: 1244.0 thalers`

Sample input: `100000`

Expected output: `The tax is: 19470.0 thalers`

Sample input: `1000`

Expected output: `The tax is: 0.0 thalers`

Sample input: `-100`

Expected output: `The tax is: 0.0 thalers`

Laboratory Activity # 4

Objectives

Familiarize the student with:

- using the `if-elif-else` statement;
- finding the proper implementation of verbally defined rules;
- testing code using sample input and output.

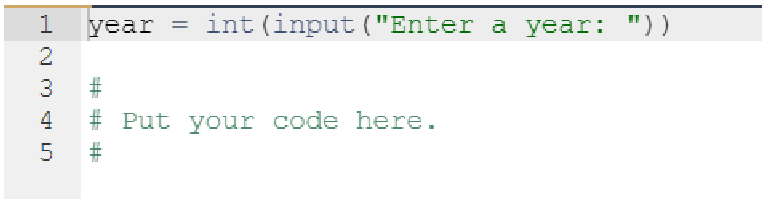
Scenario

As you surely know, due to some astronomical reasons, years may be *leap* or *common*. The former are 366 days long, while the latter are 365 days long.

Since the introduction of the Gregorian calendar (in 1582), the following rule is used to determine the kind of year:

- if the year number isn't divisible by four, it's a *common year*;
- otherwise, if the year number isn't divisible by 100, it's a *leap year*;
- otherwise, if the year number isn't divisible by 400, it's a *common year*;
- otherwise, it's a *leap year*.

Look at the code in the editor - it only reads a year number, and needs to be completed with the instructions implementing the test we've just described.

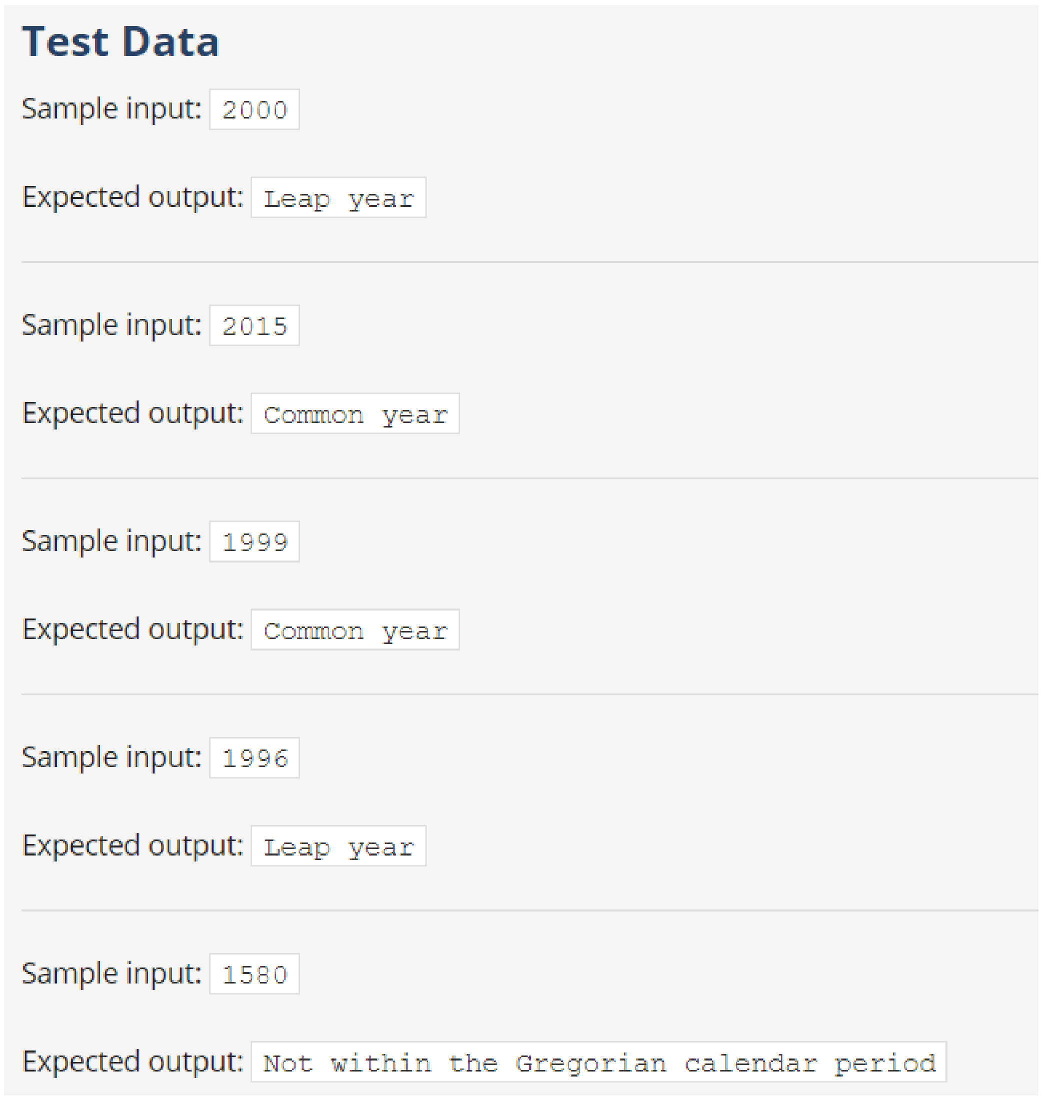


```
year = int(input("Enter a year: "))

#
# Put your code here.
#
```

The code should output one of two possible messages, which are `Leap year` or `Common year`, depending on the value entered.

It would be good to verify if the entered year falls into the Gregorian era, and output a warning otherwise: `Not within the Gregorian calendar period`. Tip: use the `!=` and `%` operators.

Test your code using the data we've provided.



Test Data

Sample input: `2000`

Expected output: `Leap year`

Sample input: `2015`

Expected output: `Common year`

Sample input: `1999`

Expected output: `Common year`

Sample input: `1996`

Expected output: `Leap year`

Sample input: `1580`

Expected output: `Not within the Gregorian calendar period`

Laboratory Activity # 5

Objectives

Familiarize the student with:

- using the `while` loop;
- reflecting real-life situations in computer code.

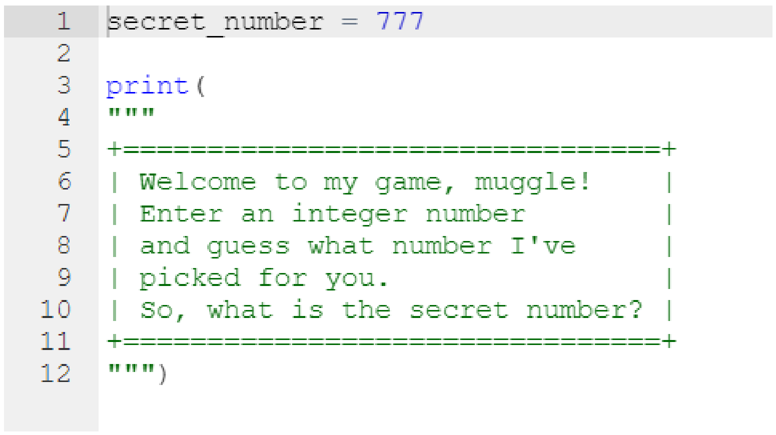
Scenario

A junior magician has picked a secret number. He has hidden it in a variable named `secret_number`. He wants everyone who run his program to play the *Guess the secret number* game, and guess what number he has picked for them. Those who don't guess the number will be stuck in an endless loop forever! Unfortunately, he does not know how to complete the code.

Your task is to help the magician complete the code in the editor in such a way so that the code:

- will ask the user to enter an integer number;
- will use a `while` loop;
- will check whether the number entered by the user is the same as the number picked by the magician. If the number chosen by the user is different than the magician's secret number, the user should see the message `"Ha ha! You're stuck in my loop!"` and be prompted to enter a number again. If the number entered by the user matches the number picked by the magician, the number should be printed to the screen, and the magician should say the following words: `"Well done, muggle! You are free now."`

The magician is counting on you! Don't disappoint him.



```
secret_number = 777

print(
"""
+================================+
| Welcome to my game, muggle!    |
| Enter an integer number        |
| and guess what number I've     |
| picked for you.                |
| So, what is the secret number? |
+================================+
""")
```

EXTRA INFO

By the way, look at the `print()` function. The way we've used it here is called *multi-line printing*. You can use **triple quotes** to print strings on multiple lines in order to make text easier to read, or create a special text-based design. Experiment with it.

Laboratory Activity # 6

Objectives

Familiarize the student with:

- using the `for` loop;
- reflecting real-life situations in computer code.

Scenario

Do you know what Mississippi is? Well, it's the name of one of the states and rivers in the United States. The Mississippi River is about 2,340 miles long, which makes it the second longest river in the United States (the longest being the Missouri River). It's so long that a single drop of water needs 90 days to travel its entire length!

The word *Mississippi* is also used for a slightly different purpose: to *count mississippily*.

If you're not familiar with the phrase, we're here to explain to you what it means: it's used to count seconds.

The idea behind it is that adding the word *Mississippi* to a number when counting seconds aloud makes them sound closer to clock-time, and therefore "one Mississippi, two Mississippi, three Mississippi" will take approximately an actual three seconds of time! It's often used by children playing hide-and-seek to make sure the seeker does an honest count.

Your task is very simple here: write a program that uses a `for` loop to "count mississippily" to five. Having counted to five, the program should print to the screen the final message `"Ready or not, here I come!"`

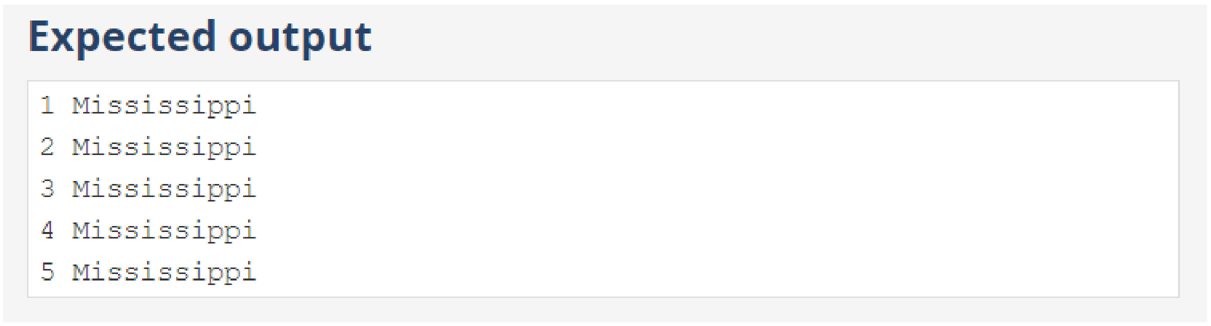
Use the skeleton we've provided in the editor.

```
import time

# Write a for loop that counts to five.
    # Body of the loop - print the loop iteration number and the word "Mississippi".
    # Body of the loop - use: time.sleep(1)

# Write a print function with the final message.
```

EXTRA INFO

Note that the code in the editor contains two elements which may not be fully clear to you at this moment: the `import time` statement, and the `sleep()` method. We're going to talk about them soon.

For the time being, we'd just like you to know that we've imported the `time` module and used the `sleep()` method to suspend the execution of each subsequent `print()` function inside the `for` loop for one second, so that the message outputted to the console resembles an actual counting. Don't worry - you'll soon learn more about modules and methods.



Expected output

```
1 Mississippi
2 Mississippi
3 Mississippi
4 Mississippi
5 Mississippi
```

Laboratory Activity # 7

Objectives

Familiarize the student with:

- using the `break` statement in loops;
- reflecting real-life situations in computer code.

Scenario

The `break` statement is used to exit/terminate a loop.

Design a program that uses a `while` loop and continuously asks the user to enter a word unless the user enters `"pasensya ka na ha? God bless!"` as the secret exit word, in which case the message `"You've successfully left the loop."` should be printed to the screen, and the loop should terminate.

Don't print any of the words entered by the user. Use the concept of conditional execution and the `break` statement.

Laboratory Activity # 8

Objectives

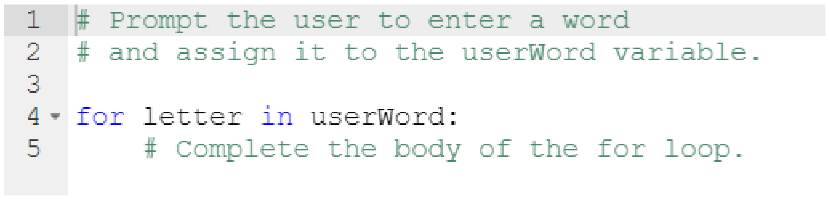
Familiarize the student with:

- using the `continue` statement in loops;
- reflecting real-life situations in computer code.

Scenario

The `continue` statement is used to skip the current block and move ahead to the next iteration, without executing the statements inside the loop.

It can be used with both the `while` and `for` loops.



```
# Prompt the user to enter a word
# and assign it to the userWord variable.

for letter in userWord:
    # Complete the body of the for loop.
```

Your task here is very special: you must design a vowel eater! Write a program that uses:

- a `for` loop;
- the concept of conditional execution (*if-elif-else*)
- the `continue` statement.

Your program must:

- ask the user to enter a word;
- use `userWord = userWord.upper()` to convert the word entered by the user to upper case; we'll talk about the so-called **string methods** and the `upper()` method very soon - don't worry;
- use conditional execution and the `continue` statement to "eat" the following vowels *A*, *E*, *I*, *O*, *U* from the inputted word;
- print the uneaten letters to the screen, each one of them on a separate line.

Test your program with the data we've provided for you.

Test data

Sample input: `Gregory`

Expected output:

```
G
R
G
R
Y
```

Sample input: `abstemious`

Expected output:

```
B
S
T
M
S
```

Laboratory Activity # 9

Objectives

Familiarize the student with:

- using the `continue` statement in loops;
- modifying and upgrading the existing code;
- reflecting real-life situations in computer code.

Scenario

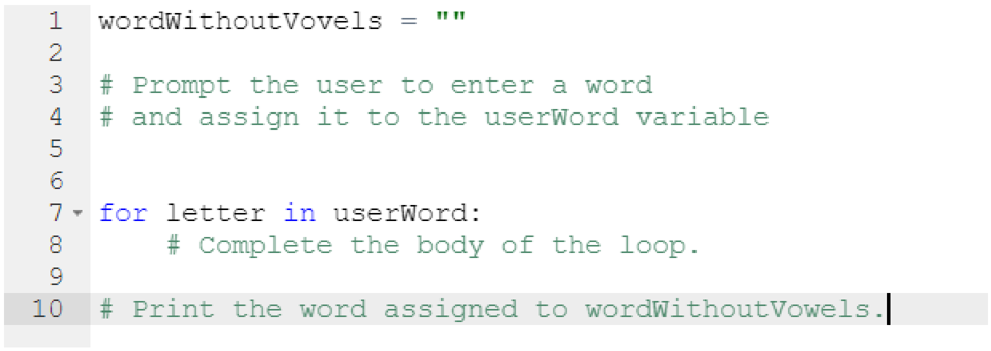
Your task here is even more special than before: you must redesign the (ugly) vowel eater from the previous lab (3.1.2.10) and create a better, upgraded (pretty) vowel eater! Write a program that uses:

- a `for` loop;
- the concept of conditional execution (*if-elif-else*)
- the `continue` statement.

Your program must:

- ask the user to enter a word;
- use `userWord = userWord.upper()` to convert the word entered by the user to upper case; we'll talk about the so-called **string methods** and the `upper()` method very soon - don't worry;
- use conditional execution and the `continue` statement to "eat" the following vowels *A*, *E*, *I*, *O*, *U* from the inputted word;
- assign the uneaten letters to the `wordWithoutVovels` variable and print the variable to the screen.

Look at the code in the editor. We've created `wordWithoutVovels` and assigned an empty string to it. Use concatenation operation to ask Python to combine selected letters into a longer string during subsequent loop turns, and assign it to the `wordWithoutVovels` variable.



```
wordWithoutVovels = ""

# Prompt the user to enter a word
# and assign it to the userWord variable


for letter in userWord:
    # Complete the body of the loop.

# Print the word assigned to wordWithoutVowels.
```

Test your program with the data we've provided for you.

Test data

Sample input: `Gregory`

Expected output:

```
GRGRY
```

Sample input: `abstemious`

Expected output:

```
BSTMS
```

Laboratory Activity # 10

Objectives

Familiarize the student with:

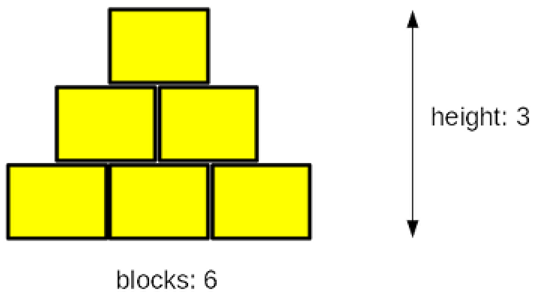
- using the `while` loop;
- finding the proper implementation of verbally defined rules;
- reflecting real-life situations in computer code.

Scenario

Listen to this story: a boy and his father, a computer programmer, are playing with wooden blocks. They are building a pyramid.

Their pyramid is a bit weird, as it is actually a pyramid-shaped wall - it's flat. The pyramid is stacked according to one simple principle: each lower layer contains one block more than the layer above.

The figure illustrates the rule used by the builders:

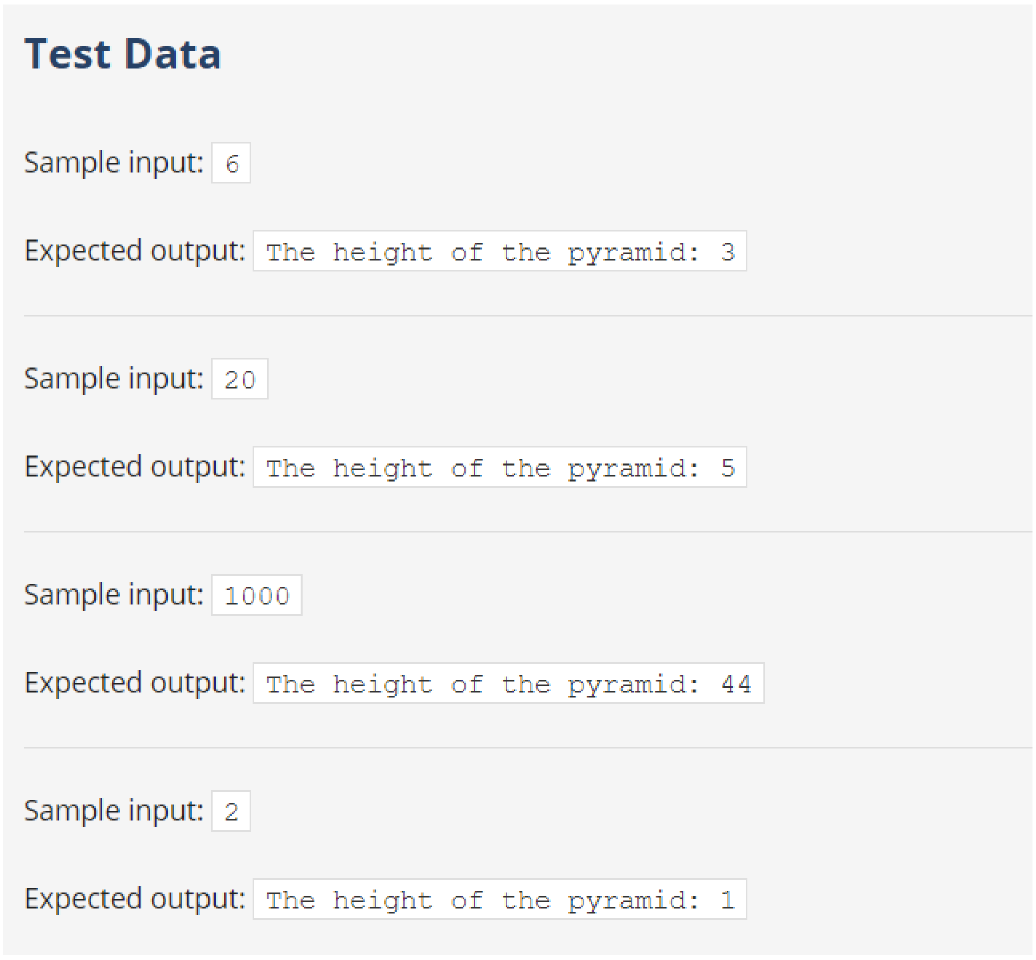


Your task is to write a program which reads the number of blocks the builders have, and outputs the height of the pyramid that can be built using these blocks.

```python
blocks = int(input("Enter the number of blocks: "))

#
# Write your code here.
#

print("The height of the pyramid:", height)
```

Note: the height is measured by the number of **fully completed layers** - if the builders don't have a sufficient number of blocks and cannot complete the next layer, they finish their work immediately.

Test your code using the data we've provided.



Test Data

Sample input: `6`

Expected output: `The height of the pyramid: 3`

Sample input: `20`

Expected output: `The height of the pyramid: 5`

Sample input: `1000`

Expected output: `The height of the pyramid: 44`

Sample input: `2`

Expected output: `The height of the pyramid: 1`

Laboratory Activity # 11

Objectives

Familiarize the student with:

- using the `while` loop;
- converting verbally defined loops into actual Python code.

Scenario

In 1937, a German mathematician named Lothar Collatz formulated an intriguing hypothesis (it still remains unproven) which can be described in the following way:

1. take any non-negative and non-zero integer number and name it `c0`;
2. if it's even, evaluate a new `c0` as `c0 ÷ 2`;
3. otherwise, if it's odd, evaluate a new `c0` as `3 × c0 + 1`;
4. if `c0 ≠ 1`, skip to point 2.

The hypothesis says that regardless of the initial value of `c0`, it will always go to 1.

Of course, it's an extremely complex task to use a computer in order to prove the hypothesis for any natural number (it may even require artificial intelligence), but you can use Python to check some individual numbers. Maybe you'll even find the one which would disprove the hypothesis.

Write a program which reads one natural number and executes the above steps as long as `c0` remains different from 1. We also want you to count the steps needed to achieve the goal. Your code should output all the intermediate values of `c0`, too.

Hint: the most important part of the problem is how to transform Collatz's idea into a `while` loop - this is the key to success.

Test your code using the data we've provided.

Test Data

Sample input: `15`

Expected output:

```
46
23
70
35
106
53
160
80
40
20
10
5
16
8
4
2
1
steps = 17
```

Sample input: `16`

Expected output:

```
8
4
2
1
steps = 4
```

Laboratory Activity # 12

Objectives

Familiarize the student with:

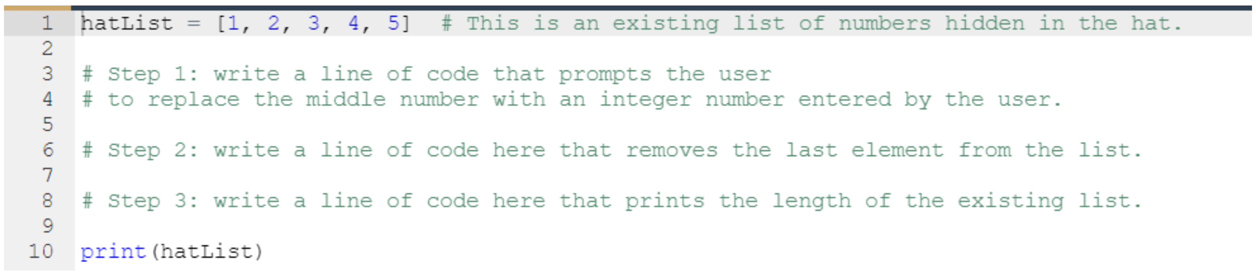
- using basic instructions related to lists;
- creating and modifying lists.

Scenario

There once was a hat. The hat contained no rabbit, but a list of five numbers: `1`, `2`, `3`, `4`, and `5`.

Your task is to:

- write a line of code that prompts the user to replace the middle number in the list with an integer number entered by the user (step 1)
- write a line of code that removes the last element from the list (step 2)
- write a line of code that prints the length of the existing list (step 3.)



```python
hatList = [1, 2, 3, 4, 5]  # This is an existing list of numbers hidden in the hat.

# Step 1: write a line of code that prompts the user
# to replace the middle number with an integer number entered by the user.

# Step 2: write a line of code here that removes the last element from the list.

# Step 3: write a line of code here that prints the length of the existing list.

print(hatList)
```

Laboratory Activity # 13

Objectives

Familiarize the student with:

- creating and modifying simple lists;
- using methods to modify lists.

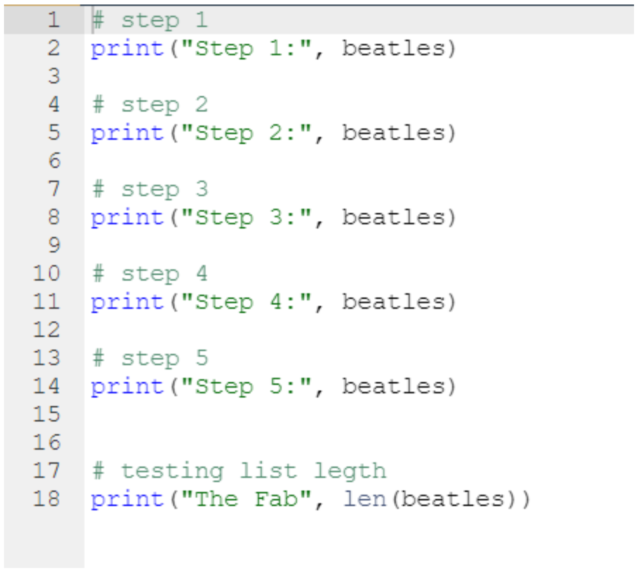
Scenario

The Beatles were one of the most popular music group of the 1960s, and the best-selling band in history. Some people consider them to be the most influential act of the rock era. Indeed, they were included in *Time* magazine's compilation of the 20th Century's 100 most influential people.

The band underwent many line-up changes, culminating in 1962 with the line-up of John Lennon, Paul McCartney, George Harrison, and Richard Starkey (better known as Ringo Starr).

Write a program that reflects these changes and lets you practice with the concept of lists. Your task is to:

- step 1: create an empty list named `beatles`;
- step 2: use the `append()` method to add the following members of the band to the list: `John Lennon`, `Paul McCartney`, and `George Harrison`;
- step 3: use the `for` loop and the `append()` method to prompt the user to add the following members of the band to the list: `Stu Sutcliffe`, and `Pete Best`;
- step 4: use the `del` instruction to remove `Stu Sutcliffe` and `Pete Best` from the list;
- step 5: use the `insert()` method to add `Ringo Starr` to the beginning of the list.



```
# step 1
print("Step 1:", beatles)

# step 2
print("Step 2:", beatles)

# step 3
print("Step 3:", beatles)

# step 4
print("Step 4:", beatles)

# step 5
print("Step 5:", beatles)


# testing list legth
print("The Fab", len(beatles))
```

Laboratory Activity # 14

Objectives

Familiarize the student with:

- list indexing;
- utilizing the `in` and `not in` operators.

Scenario

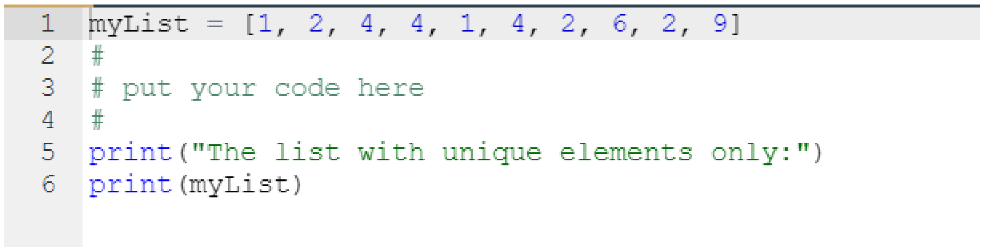
Imagine a list - not very long, not very complicated, just a simple list containing some integer numbers. Some of these numbers may be repeated, and this is the clue. We don't want any repetitions. We want them to be removed.

Your task is to write a program which removes all the number repetitions from the list. The goal is to have a list in which all the numbers appear not more than once.

Note: assume that the source list is hard-coded inside the code - you don't have to enter it from the keyboard. Of course, you can improve the code and add a part that can carry out a conversation with the user and obtain all the data from her/him.

Hint: we encourage you to create a new list as a temporary work area - you don't need to update the list in situ.

We've provided no test data, as that would be too easy. You can use our skeleton instead.



```python
myList = [1, 2, 4, 4, 1, 4, 2, 6, 2, 9]
#
# put your code here
#
print("The list with unique elements only:")
print(myList)
```