**PRACTICUM REPORT**

**ALGORITHM AND DATA STRUCTURES**

**MODUL 1 : PYTHON REVIEW**



**Disusun Oleh :**

**JOHN DOE**

**L2002XXXXX**

**X**

**INFORMATICS ENGINEERING**

**FACULTY OF COMMUNICATION AND INFORMATICS**

**UNIVERSITAS MUHAMMADIYAH SURAKARTA**

**YEARS 2024/2025**

**1.11 Questions**

1. Create a function cetakSiku(x) that will print the following:

\*

\*\*

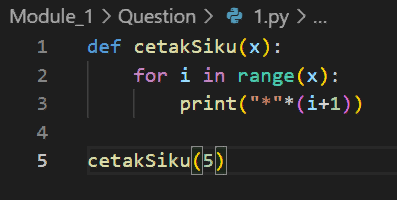
\*\*\*

\*\*\*\*

\*\*\*\*\*

The x value shows the height of the triangle (the image above means it can be obtained from running cetakSiku(5)). Use a double loop!

* **Program Code**



Picture 1.1 *the code.*

* **Practicum Result Screenshot**

****

Gambar 1.2 the output

1. Create a function that accepts two positive integers, whisch will draw a rectangular shape. Example of calling :  
   >>> gambarlahPersegiEmpat(4,5) # Button <enter>

Presed

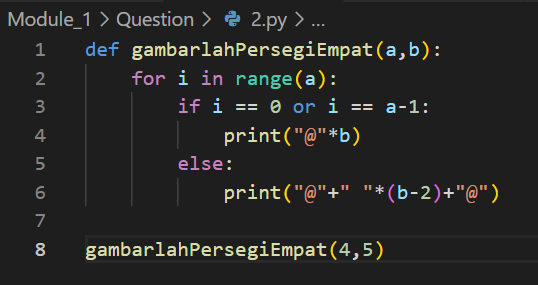
@@@@@

@ @

@ @

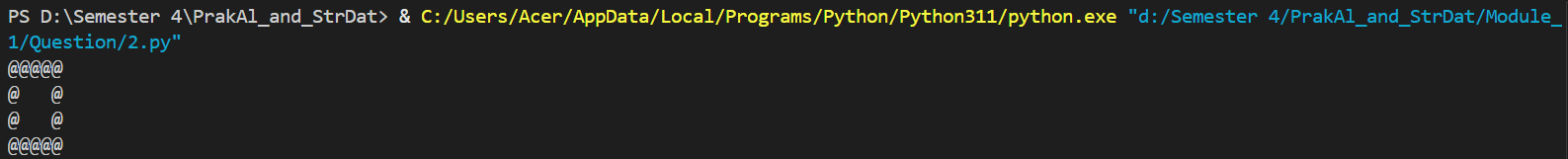
@@@@@

* **Program Code**



Picture 2.1 the code.

* **Practikum result screenshot**



Picture 2.2 the output.

1. The following are two interrelated questions
2. Create a function that accepts a string and returns a list of two integers. These two returned integers are: the number of letters in that string and the number of vowels (vowels are vowels) in that string. Example of calling:

>>> k = jumlahHurufVokal(’Surakarta’)

>>> k

(9, 4)# Nine letters, and four of them are

Vowels

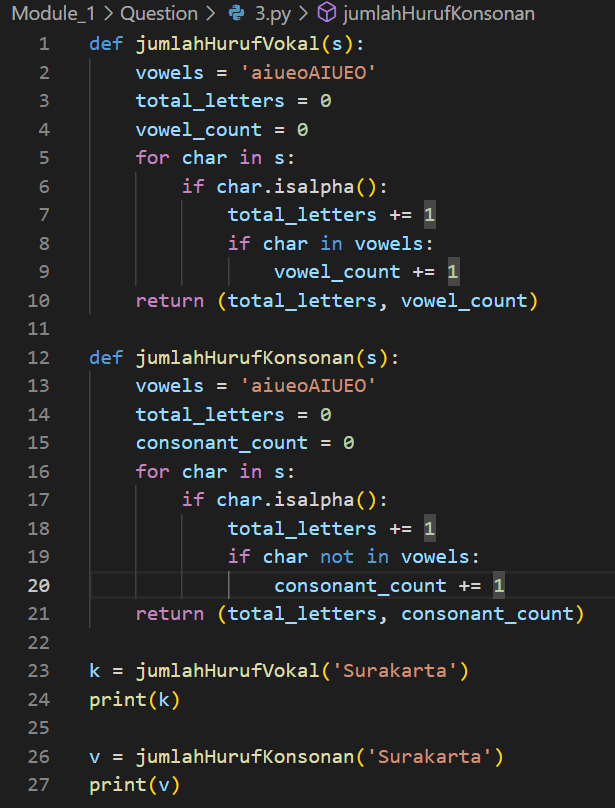
1. Same as question (a) above, but now the consonants are counted. There's only one different line in the code! Example of calling:

>>> k = jumlahHurufKonsonan(’Surakarta’)

>>> k

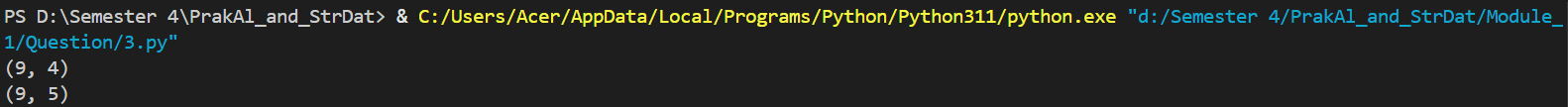
(9, 5)# Nine

* **Program code**



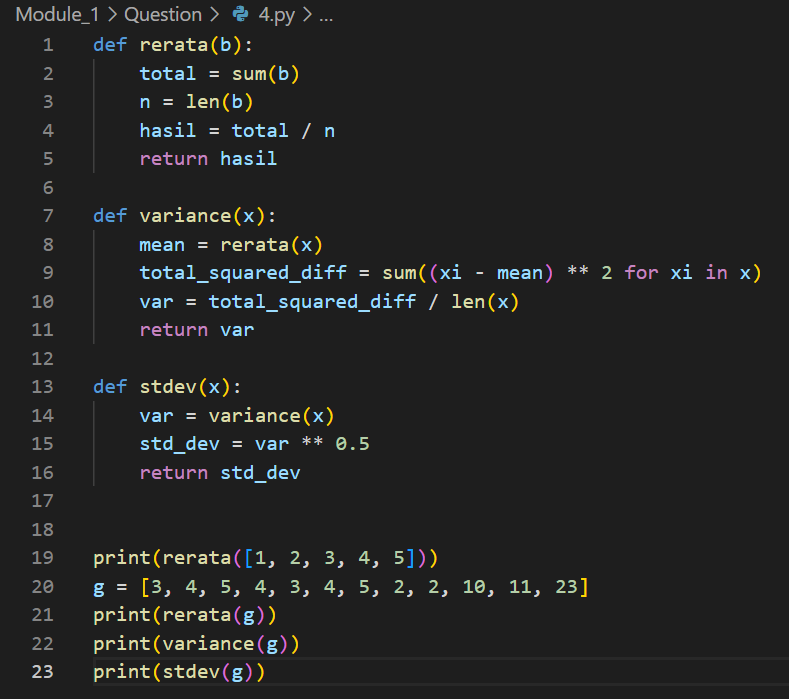
Picture 3.2 the code

* **Practikum results screenshot**



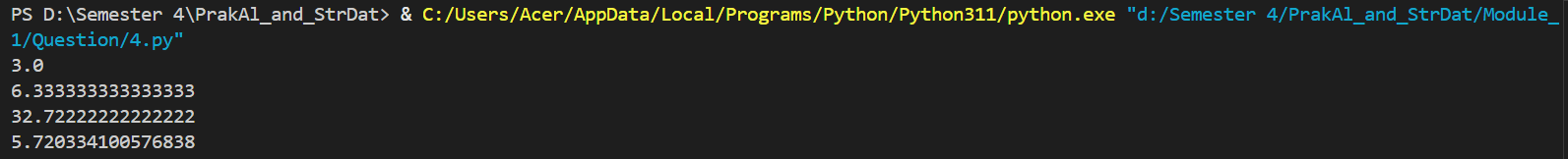
|  |
| --- |
| Create a function that calculates the average of an array containingnumbers. The average has a formula  𝑥̅=  ∑ 𝑛 𝑖=1 𝑥𝑖  𝑛  (1.3)  But remember that Python starts index from 0. The function must have a form rerata(x), where x is a list containing the numbers whose average you want to calculate. So, your work will have a form:  • Create a file with contents like this  • Run the program by pressing “F5”, then call the program like  this  rerata([1,2,3,4,5])#hasilnya 3  g = [3,4,5,4,3,4,5,2,2,10,11,23]  rerata(g)  *Extra credit: Also create a function to calculate the variance and*  *standard deviation with the prototype, respectively*, variance(x)  *and* stdev(x). |

* **Program code**



Picture 4.1 the code

* **Practicum results screenshot**



Picture 4.2 the output

1. Create a function to determine whether an integer is a prime number or not. To make it easier, complete the program below Once done, run the above program and then test it in Python Shell:

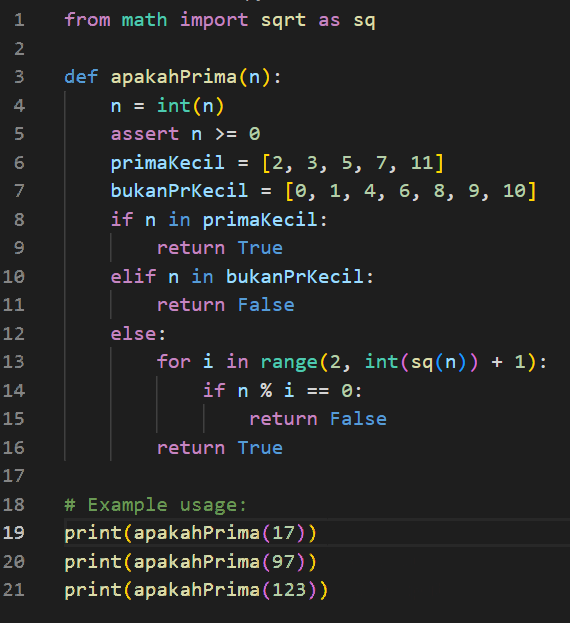
|  |
| --- |
| from math import sqrt as sq  def apakahPrima(n):  n =int(n)# If the number is a fraction,  discard the fraction.  assert n>=0 # Only accepts non-negative  numbers.  primaKecil = [2,3,5,7,11] #If the number is  small, then  bukanPrKecil = [0,1,4,6,8,9,10] #caugh here.  if n in primaKecil:  return True  elif n in bukanPrKecil:  return False  else:  for I in range(2,int(sq(n))+1): # Just  get to the roots.  .....#Your task  .....#is fill  .....# fill in this dot. |

apakahPrima(17)

apakahPrima(97)

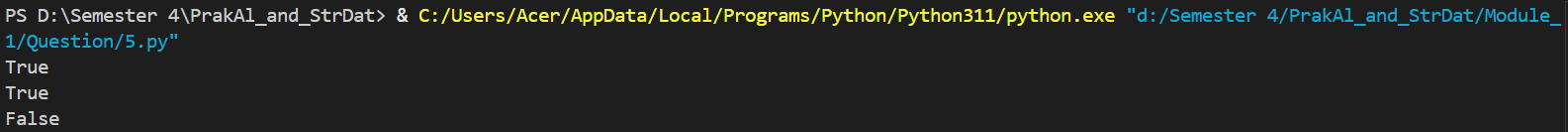
apakahPrima(123)

* **Program code**



Picture 5.1 the code

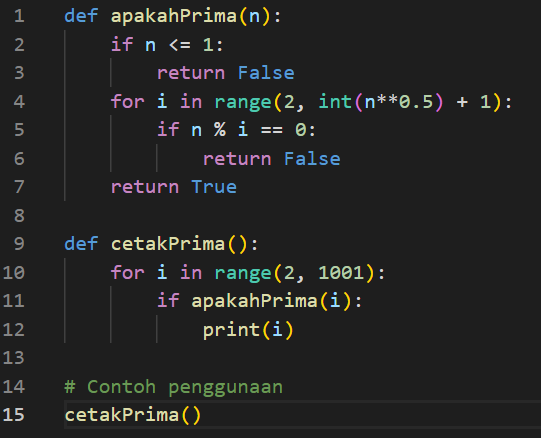
* **Practicum result screenshot**



Picture 5.2 the output.

1. Write a program that prints all the prime numbers from 2 to 1000. Make use of functions **apakahPrima()** on the number above.

* **Program code**



Picture 6.1 the code

* **Practicum results screenshot**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Picture 6.2 the output

1. Write a program that accepts a positive integer and provides its

prime factorization. Prime factorization is factoring an integer into

its constituent prime numbers. Example:

>>> faktorPrima(10)

(2, 5)

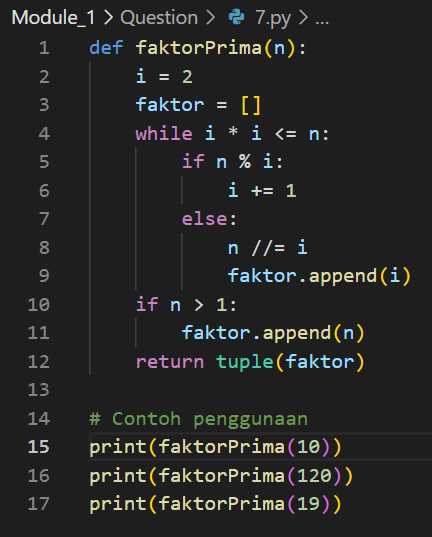
>>> faktorPrima(120)

(2, 2, 2, 3, 5)

>>> faktorPrima(19)

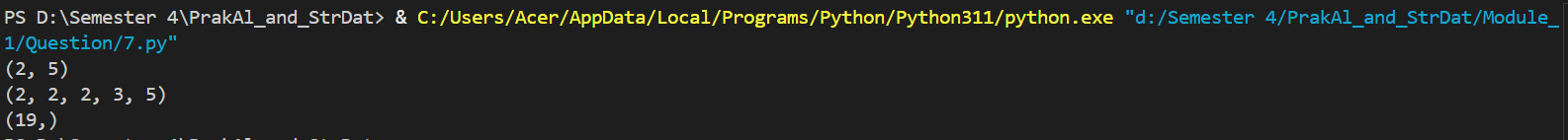
(19,)

* **Program code**



Picture 7.1 the code

* **Practicum results screenshot**



Picture 7.2 the output.

1. Create a function **apakahTerkandung(a,b)** which accepts two

strings **a** and **b**, then determines whether string a is contained in

string b. The execution is like this:

>>> h = ’do’

>>> k = ’Indonesia tanah air beta’

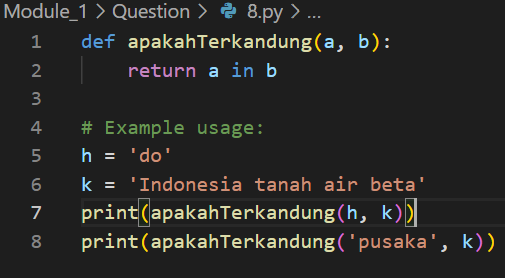
>>> apakahTerkandung(h,k)

True

>>> apakahTerkandung(’pusaka’,k)

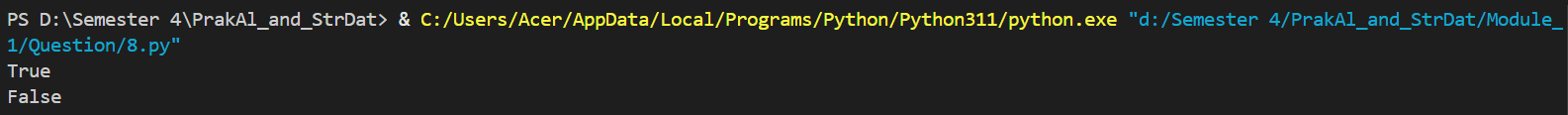
False

* **Program code**



Picture 8.1 the code.

* **Practicum results screenshot**



Picture 8.1 the output.

1. Create a program to print numbers from 1 to 100. If the number is a

multiple of 3, print 'Python'. If it is a multiple of 5, print 'UMS'. If it

fits multiples of 3 and multiples of 5, print 'Python UMS'. So the

result:

1

2

Python

4

UMS

Python

7

8

Python

UMS

11

Python

13

14

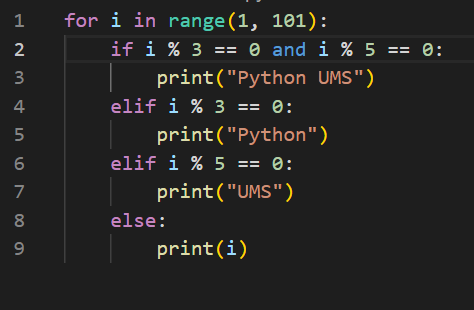
Python UMS

16

17

...

* **Program code**



Picture 9.1 the code

* **Practicum results screensho**

|  |  |  |
| --- | --- | --- |
|  |  |  |

Picture 9.2 the output.

1. Make a modification of Example 1.4, to capture the case where the

determinant is less than zero. If this happens, display an on-screen

warning like this:

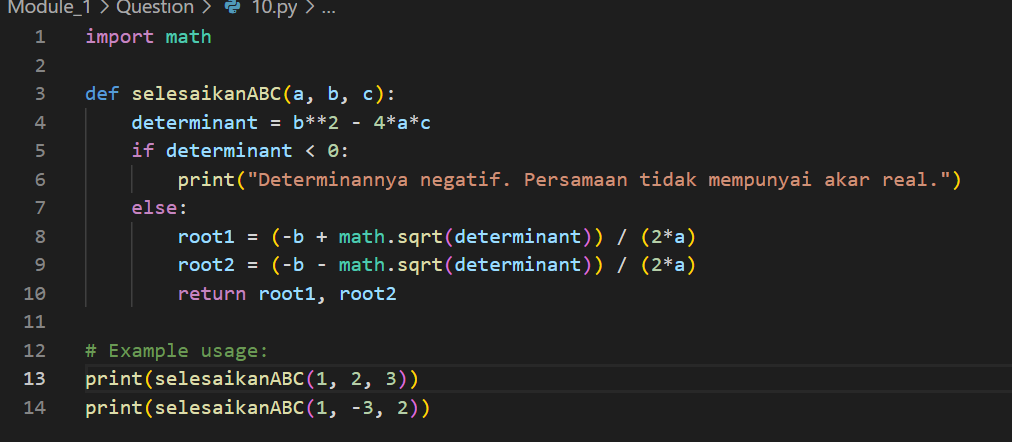
>>> selesaikanABC(1,2,3)

Determinannya negatif. Persamaan tidak mempunyai

akar real.

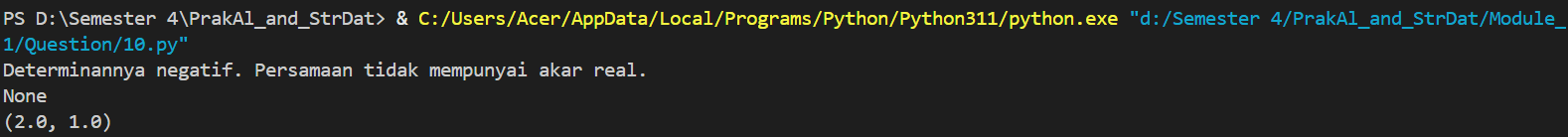
>>>

* **Program code**



Picture 10.1 the code.

* **Practicum results**



Picture 10.2 the output.

1. Create a function **apakahKabisat()** yang menerima suatu angka

(tahun). which receives a number (year). If the year is a leap year,

return it **True**. If it's not leap, return it **False**.

A leap year – a year with a date of February 29 – is a year that is

divisible by 4, unless it is divisible by 100 (in which case it is not a

leap year). But if it is divisible by 400, it is a leap year (even if it is

divisible by 100).

The following are some examples:

• 1896 leap year (divisible by 4)

• 1897 was not a leap year (obviously)

• 1900 is not a leap year (even though it is divisible by 4, it is

divisible by 100, and not divisible by 400)

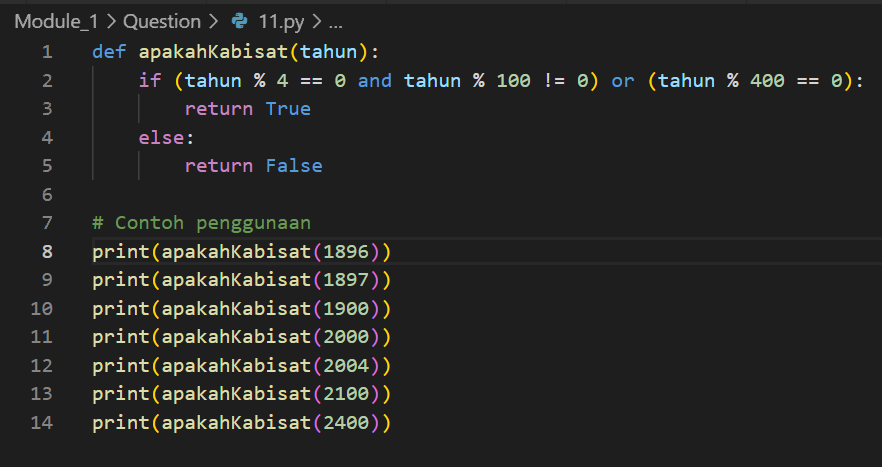
• 2000 leap years (divisible by 400)

• 2004, 2008, 2012, 2016, ..., 2096 leap year

• 2100, 2200, 2300 are not leap years

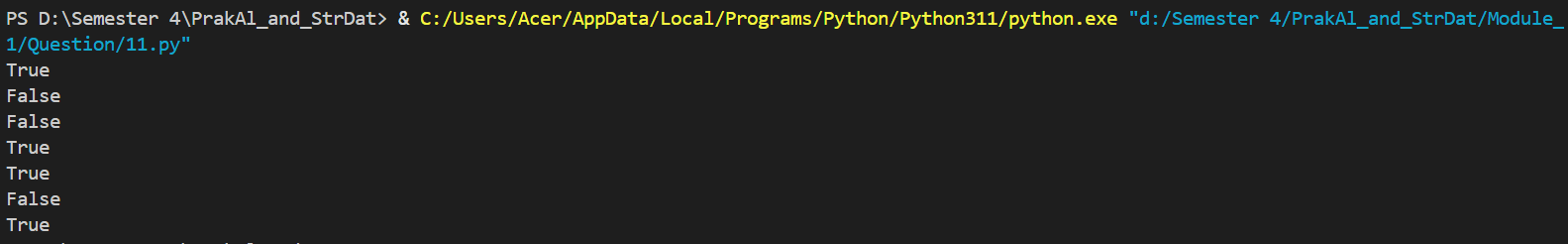
• 2400 leap years

* **Program code**



Picture 11.1 the code.

* **Practicum results**



Picture 11.2 the results

1. Number guessing game program. Create a program whose global

flow is like this:

• The computer generates a random integer between 1 and 100.

The value is stored in a variable and is not displayed to the user.

• The user is asked to guess the number, entered via the keyboard.

• If the input number is too small or too large, the user gets

feedback from the computer (“That number is too small. Try

again”)

• The process is repeated until the number is guessed or until a

certain number of guesses are wrong 8 .

When the program is run, the process is more or less like below

Number guessing game.

I store a round number between 1 and 100. Guess

what.

Enter 1st guess: > 50

It's too small. Try again.

Enter the 2nd guess: > 75

It's too big. Try again.

Enter the 3rd guess: > 58

1920

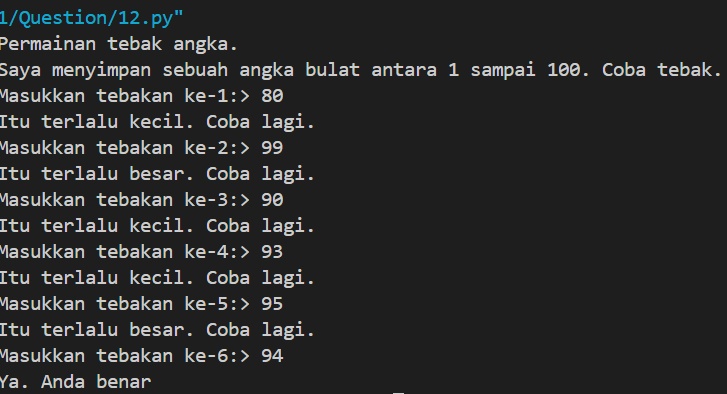
Yes. You are right

* **Program code**



Picture 12.1 the code.

* **Practicum results**



Picture 12.2 the results

1. Creat a function **katakan()** which accepts a positive integer and

returns a string which is the Indonesian pronunciation of that

number. Example:

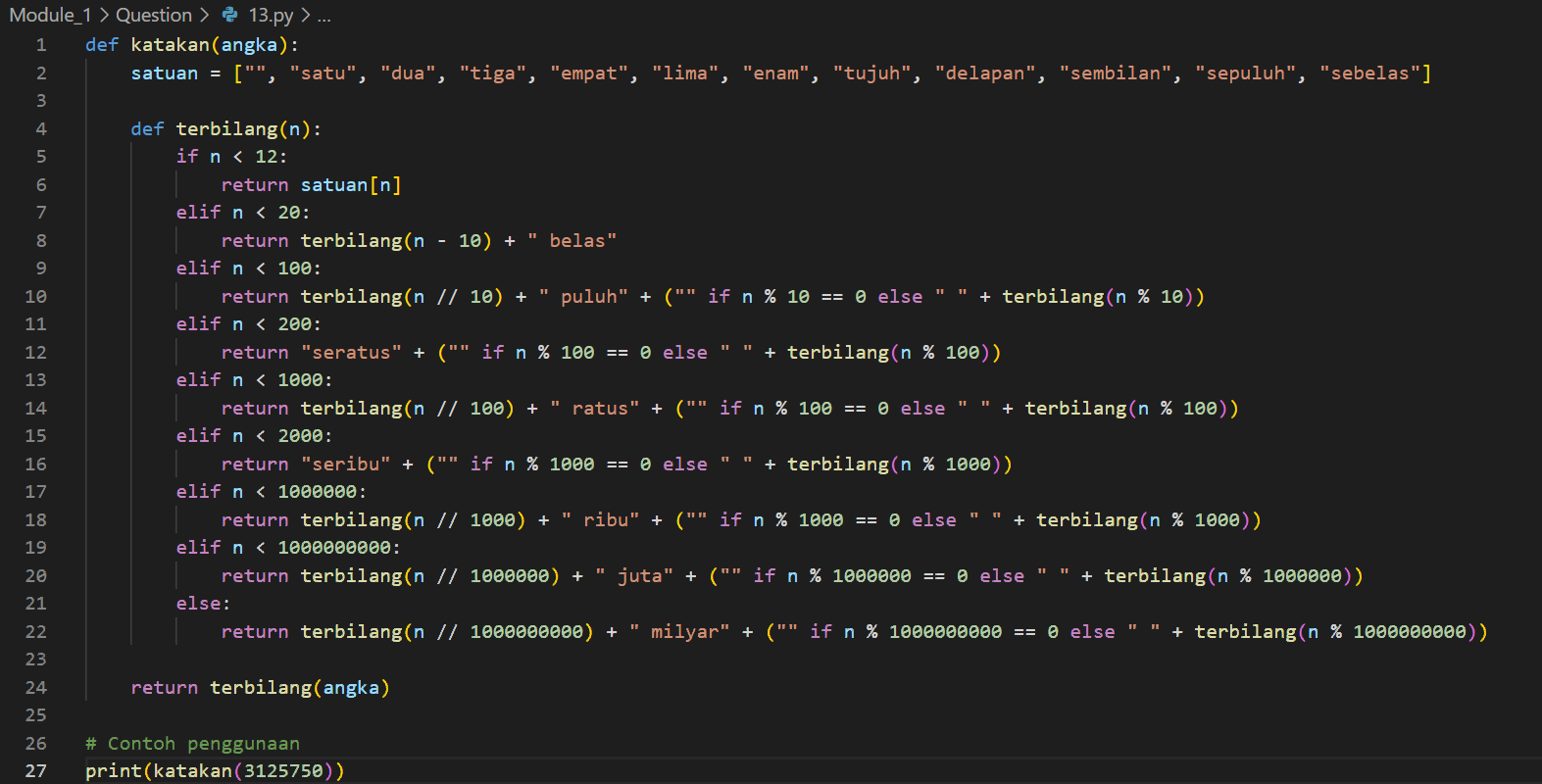
>>> katakan(3125750)

’Tiga juta serratus dua puluh lima ribu tujuh ratus

lima puluh’

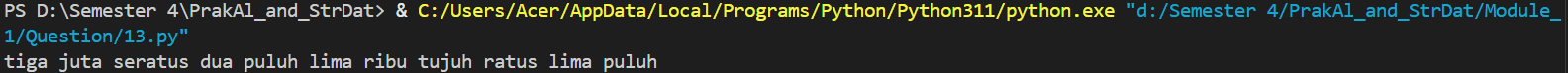
Limit the input to less than one billion. Extra credit: use recursion.

* **Program code**



Picture 13.1 the code.

* **Practicum results**



Picture 13.2 the output.

1. Creat a function **formatRupiah()** which accepts a positive integer

and returns a string which is that number but in 'rupiah format'.

Example:

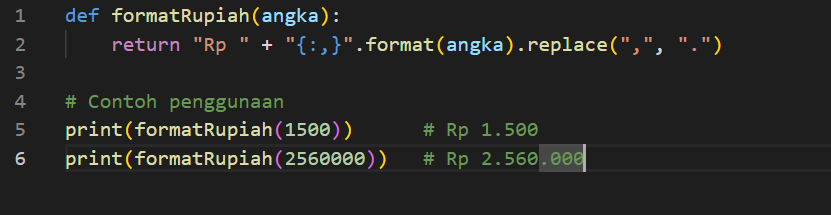
>>> formatRupiah(1500)

’Rp1.500’

>>> formatRupiah(2560000)

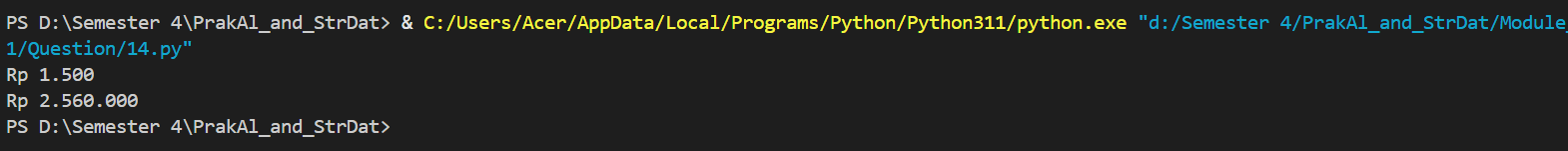
’Rp2.560.000’

* **Program code**



Picture 14.1 the code

* **Practicum results**



Picture 14.2 the resu;lts