Modul 5

Modul, Package, String dan Exception

youtube.com/isnaalfi

Module

Kenapa perlu modul?

- Program terus berkembang
- Kadang membutuhkan code yang pernah dibuat

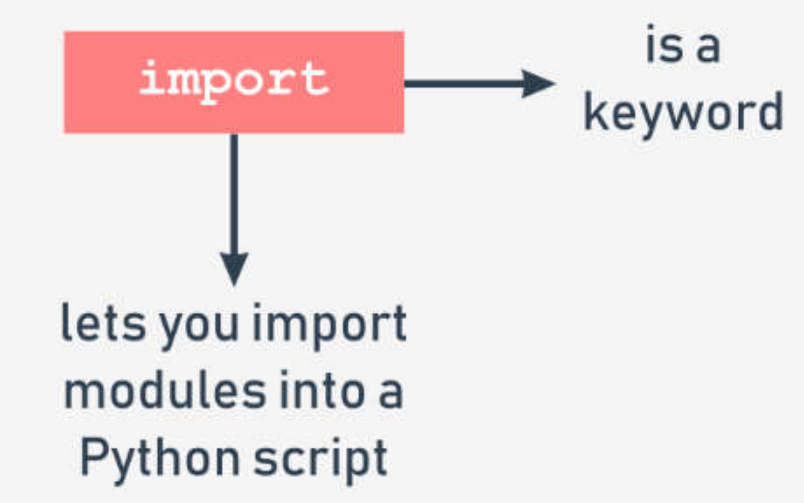
Bagaimana menggunakan modul?

Importing a module

Importing a module is done by an instruction named `import`

the clause contains:

- the `import` keyword;
- the `name` of the module which is subject to import.

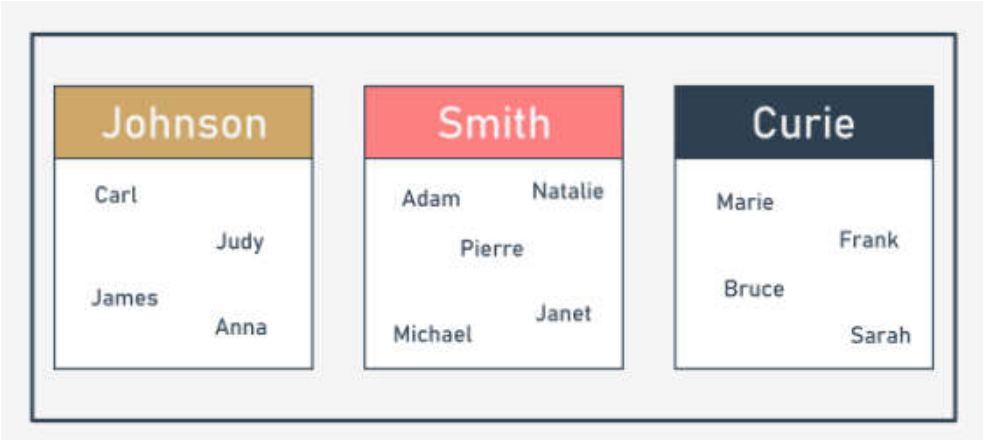


In [44]:
```
import math
```

In [45]:
```
import math, sys
```

namespace

A namespace is a space (understood in a non-physical context) in which some names exist and the names don't conflict with each other (i.e., there are not two different objects of the same name).



In [46]:
```
import numpy
import math
import scipy

pi=200
print(math.pi)
print(numpy.pi)
print(scipy.pi)
print(pi)


#note: pi inside the module won't be affected by pi that we declare in main prog
ram
```

```
3.141592653589793
3.141592653589793
3.141592653589793
200
```

In [47]:
```
from math import pi

print(pi)
print(e)
```

```
3.141592653589793

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-47-da29c8d1bdf5> in <module>
      2
      3 print(pi)
----> 4 print(e)

NameError: name 'e' is not defined
```

The instruction consists of the following elements:

- the `from` keyword;
- the `name of the module` to be (selectively) imported;
- the `import` keyword;
- the `name` or `list of names` of the entity/entities which are being imported into the namespace.

In [48]:
```
## override nilai sin dan pi
from math import sin, pi

print(sin(pi/2))

pi = 3.14

def sin(x):
    if 2 * x == pi:
        return "Ini hasil fungsi buatan"
    else:
        return None

print(sin(pi/2))
```

```
1.0
Ini hasil fungsi buatan
```

Mengimport semua modul

In [ ]:
```
from module import *
```

Nama dari entitas digantikan dengan asterisk tunggal `*`

`*` merupakan instruksi untuk meng-import semua entitas yang ada

Aliasing

Untuk nama file yang akan di `import` kan dapat dilakukan proses `aliasing`

`Aliasing` menyebabkan `modul` diidentifikasi dengan `nama yang berbeda` dari `aslinya`

`import` module `as` alias

`as` merupakan kata kunci untuk melakukan `aliasing`

Jika kita ingin merename `math` , dengan `m` dapat dilakukan dengan cara sebagai berikut.

In [ ]:
```
import math as m

print(m.pi) #math.pi

print(__name__)

print(math.pi)
```

Note : after successful execution of an aliased import, the original module name becomes inaccessible and must not be used.

from `module` **import** `name` **as** `alias`

from `module` **import** `n` **as** `a` , `m` **as** `b` , `o` **as** `c`

In [ ]:
```
from math import pi as PI, sin as sine

print(sine(PI/2))
```

Working with standard modules

dir(`module`)

The function returns an alphabetically sorted list containing all entities' names available in the module

In [ ]:
```
import math

for name in dir(math):
    print(name, end="\t")
```

`math` module

Let's start with a quick preview of some of the functions provided by the math module.

The first group of the math's functions are connected with trigonometry:

- `sin(x)` → the sine of x;
- `cos(x)` → the cosine of x;
- `tan(x)` → the tangent of x.

Here are also their inversed versions:

- `asin(x)` → the arcsine of x;
- `acos(x)` → the arccosine of x;
- `atan(x)` → the arctangent of x.

`x` is a radian

These functions take one argument (mind the domains) and return a measure of an angle in radians.

To effectively operate on angle measurements, the math module provides you with the following entities:

- `pi` → a constant with a value that is an approximation of π;
- `radians(x)` → a function that converts x from degrees to radians;
- `degrees(x)` → acting in the other direction (from radians to degrees)

In [ ]:
```
from math import pi, radians, degrees, sin, cos, tan, asin

ad = 90
ar = radians(ad)
ad = degrees(ar)

print(ad == 90.)
print(ar == pi / 2.)
print(sin(ar) / cos(ar) == tan(ar))
print(asin(sin(ar)) == ar)
```

Another group of the math's functions is formed by functions which are connected with exponentiation:

- `e` → a constant with a value that is an approximation of Euler's number (e)
- `exp(x)` → finding the value of ex;
- `log(x)` → the natural logarithm of x
- `log(x, b)` → the logarithm of x to base b
- `log10(x)` → the decimal logarithm of x (more precise than log(x, 10))
- `log2(x)` → the binary logarithm of x (more precise than log(x, 2))

In [ ]:
```
from math import e, exp, log

print(pow(e, 1) == exp(log(e)))
print(pow(2, 2) == exp(2 * log(2)))
print(log(e, e) == exp(0))
```

Built-in function

Note: the pow() function:

`pow(x, y)` → finding the value of xy (mind the domains)

This is a built-in function, and doesn't have to be imported.

The last group consists of some general-purpose functions like:

- ceil(x) → the ceiling of x (the smallest integer greater than or equal to x)
- floor(x) → the floor of x (the largest integer less than or equal to x)
- trunc(x) → the value of x truncated to an integer (**be careful** - it's **not an equivalent** either of ceil or floor)
- factorial(x) → returns x! (x has to be an integral and not a negative)
- hypot(x, y) → returns the length of the hypotenuse of a right-angle triangle with the leg lengths equal to x and y (the same as sqrt(pow(x, 2) + pow(y, 2)) but more precise)

It demonstrates the fundamental differences between `ceil()`, `floor()` and `trunc()`.

In [49]:
```
from math import ceil, floor, trunc, hypot

x = 1.4
y = 2.6

print(floor(x), floor(y))
print(floor(-x), floor(-y))
print(ceil(x), ceil(y))
print(ceil(-x), ceil(-y))
print(trunc(x), trunc(y))
print(trunc(-x), trunc(-y))

a = 4
b = 3

print(hypot(a,b))
```

```
1 2
-2 -3
2 3
-1 -2
1 2
-1 -2
5.0
```

`random` Module

It delivers some mechanisms allowing you to operate with **pseudorandom numbers**.

pseudo - : the numbers generated by the modules may look random in the sense that you cannot predict their subsequent values, but don't forget that they all are calculated using very refined algorithms.

In [50]:
```
from random import random

for i in range(5):
    print(random())
```

```
0.9276064937341642
0.6165902592752007
0.9035150367343935
0.4187009735203726
0.19184526764164467
```

If you want integer random values, one of the following functions would fit better:

- randrange(end)
- randrange(beg, end)
- randrange(beg, end, step)
- randint(left, right)

In [55]:
```
from random import randrange, randint

print(randrange(10), end=' ')
print(randrange(3, 15), end=' ')
print(randrange(2, 10, 3), end=' ')
print(randint(2,5))
```

```
8 6 8 5
```

This is what we got in one of the launches:

9,4,5,4,5,8,9,4,8,4,

It's a function named in a very suggestive way - choice:

choice(sequence) sample(sequence, elements_to_choose=1)

In [60]:
```
from random import choice, sample

lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(choice(lst))
print(sample(lst, 5))
print(sample(lst, 10))
```

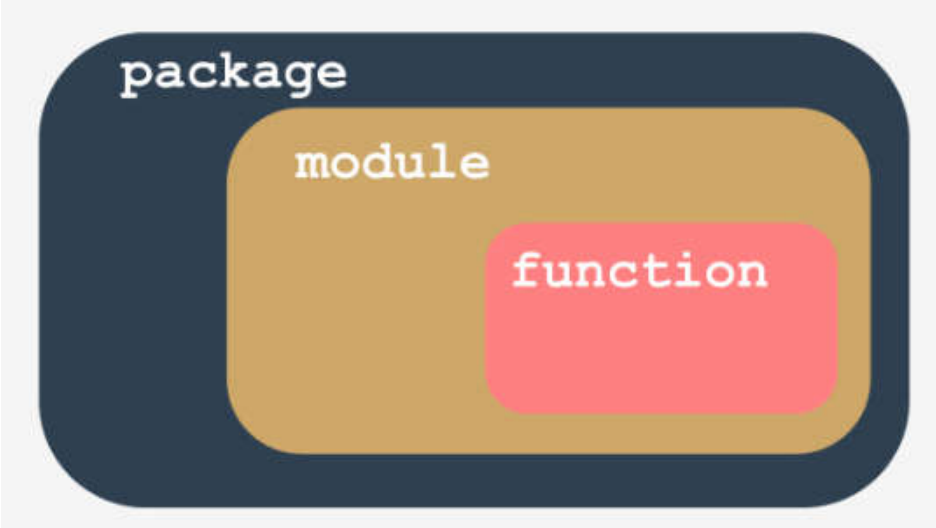
```
6
[7, 2, 6, 3, 4]
[7, 4, 3, 6, 5, 1, 8, 10, 9, 2]
```

In [1]:
```
from random import randint, random

for i in range(10):
    print(randint(1, 10), end=',')
```

```
6,7,8,3,3,5,10,5,2,2,
```

You can read about all standard Python modules here: https://docs.python.org/3/py-modindex.html (https://docs.python.org/3/py-modindex.html).

Package



- **a module is a kind of container filled with functions** - you can pack as many functions as you want into one module and distribute it across the world;
- of course, it's generally **a good idea not to mix functions with different application areas** within one module
- making many modules may cause a little mess - sooner or later you'll want to **group your modules** exactly in the same way as you've previously grouped functions
- **package**; in the world of modules, a package plays a similar role to a folder/directory in the world of files.

Membuat modul

Pertama,kita membuat 2 file dengan nama aritmatika.py dan main.py

Langkah:

artimatika.py:

- Buka python IDLE
- Klik **file** dan pilih **new file**
- Simpan file dengan nama **aritmatika.py**

main.py:

- Buka python IDLE
- Klik **file** dan pilih **new file**
- Simpan file dengan **main.py**

Note: **Kedua file disimpan dalam satu folder yang sama.**

In [ ]:
```
def tambah(a,b): # 3,4
     return a+b
def kurang(a,b):
     return a-b
def kali(a,b):
     return a*b
def bagi(a,b):
     return a/b
```

In [ ]:
```
import aritmatika
```

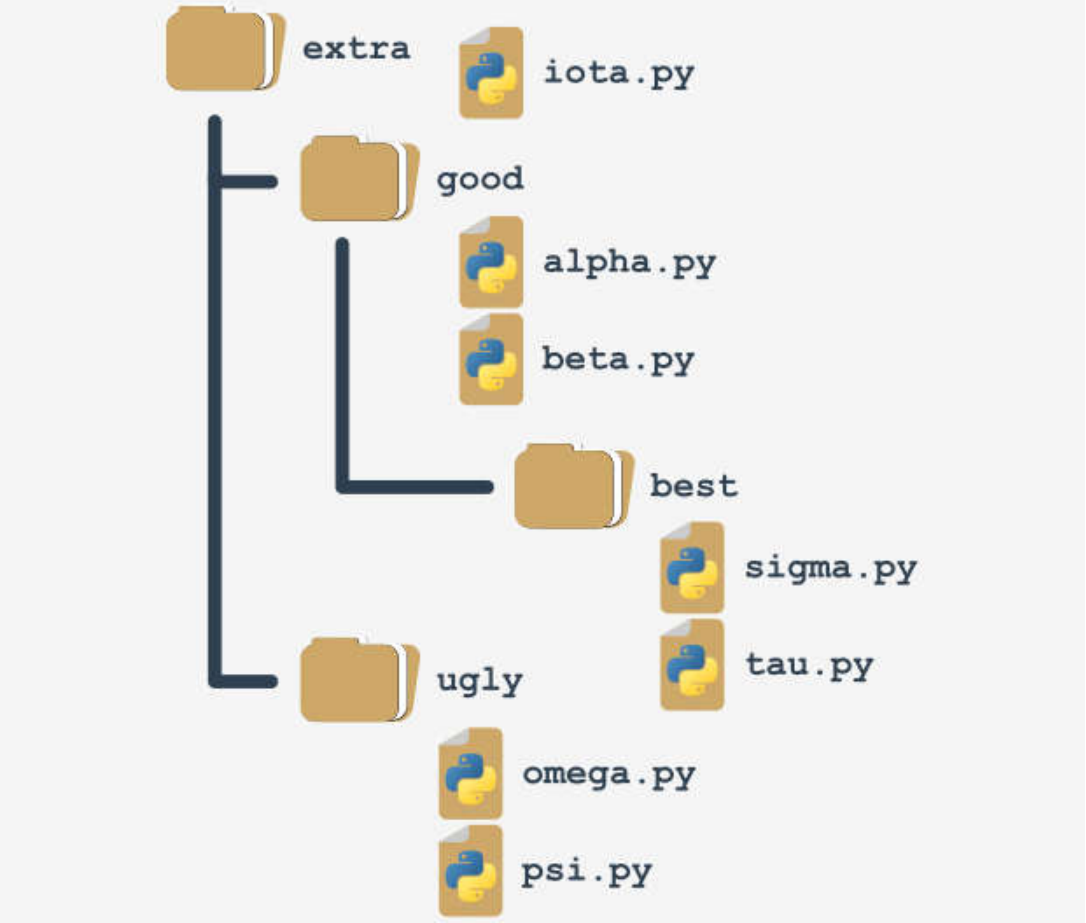
In [ ]:
```
#main.py
import aritmatika as art

a=art.tambah(3,4)   # 7
b=art.kurang(3,4)
c=art.kali(3,4)
d=art.bagi(3,4)

print(a)
print(b)
print(c)
print(d)
```

In [ ]:
```
from aritmatika import tambah

a=tambah(10,3)
print(a)

print(tambah(3,4))
```

In [ ]:
```
import math as m

m.copysign()
```

Package

Konsep package



`packages` , like modules, may require initialization.

Python expects that there is a file with a very unique name inside the package's folder: `__init__.py`.

The content of the file is executed when any of the package's modules is imported.

If you don't want any special initializations, you can leave the **file empty**, but you mustn't omit it.

1. Buatlah folder dengan nama LATIHAN_PYTHON
2. Dalam folder LATIHAN_PYTHON, buatlah
   - folder dengan nama **latihan_package**,
   - file main.py, dan
   - file `__init__.py`
3. Dalam **latihan_package**, buatlah 2 file, dengan nama
   - alpha.py
   - beta.py

In [ ]:
```
#alpha.py
def alphaSatu():
    print("alphaSatu")

def alphaDua():
    print("alphaDua")
```

In [ ]:
```
#beta.py
def betaSatu():
    print("betaSatu")

def betaDua():
    print("betaDua")
```

In [ ]:
```
#main.py

import latihan_package.alpha as a
import latihan_package.beta as b

a.alphaSatu()
b.betaDua()
```

In [61]:
```
#cara mengakses package yang dibuat, copy dan paste code dalam file main.py

import os

os.chdir("C:\\Users\\Bunda Freya\\Desktop\\LATIHAN_PYTHON")

import latihan_package.alpha as a, latihan_package.beta as b

a.alphaSatu()
b.betaSatu()
```

```
alphaSatu
betaSatu
```

In [ ]:
```
import os
os.chdir(r"E:\\")
os.getcwd()
```

In [ ]:
```
os.chdir(r"E:\CTA\DIGITAL TALENT\digital-talent\2019")

import latihan_package.alpha as a
a.alphaSatu()
```

In [ ]:
```
import alpha

alpha.alphaSatu()
```

Errors, failures, and other

Kesalahan merupakan hal yang sering terjadi dalam proses pembuatan pemrograman.

Sebab terjadinya kesalahan:

- Kesalahan dalam penulisan kode, sehingga kode tidak dapat dijalankan sama sekali
- Kesalahan yang terjadi ketika program sedang di eksekusi

Dua buah cara yang dapat digunakan untuk memeriksa suatu kesalahan:

- menggunakan blok `try....except`
- menggunakan statement `assert`

Eksepsi

Merupakan penanganan kesalahan yang dilakkan pada saat proses eksekusi program, dan yang akan mencegah alur dari perintah-perintah normal yang terdapat di dalam program.

Pada kode di bawah masih memungkinkan terjadi kesalahan, yaitu:

- user menginputkan string, dan
- user menginputkan bilangan negatif

In [62]:
```
import math

x = float(input("Enter x: "))
y = math.sqrt(x)

print("The square root of", x, "equals to", y)
```

```
Enter x: qetejewjbwe

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-62-8c0b3ffe9411> in <module>
      1 import math
      2
----> 3 x = float(input("Enter x: "))
      4 y = math.sqrt(x)
      5

ValueError: could not convert string to float: 'qetejewjbwe'
```

In [63]:
```
value = 1
value /= 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-63-ad45975c684c> in <module>
      1 value = 1
----> 2 value /= 0

ZeroDivisionError: division by zero
```

`NameError` , `ValueError` , `ZeroDvisionError` merupakan eksepsi untuk mengatasi kesalahan-kesalahan yang terjadi seperti contoh diatas

Kedua aktivitas ini disebut **memunculkan(raising) eksepsi**. Kita dapat mengatakan bahwa Python selalu memunculkan eksepsi (atau bahwa eksepsi telah dimunculkan) ketika ia tidak tahu apa yang harus dilakukan dengan kode Anda. Yang menyebabkan:

- eksepsi mengharapkan ada sebuah perintah yang dapat menangani atau mencegah terjadinya kesalahan
- jika tidak ada perintah untuk menangani atau mencegah terjadinya kesalahan tersebut, python akan menghentikan atau **terminated** program, sehingga akan muncul pesan **error**
- jika kesalahan dapat ditangani, python akan melanjutkan pada kode program selanjutnya

`try .... except`

In [64]:
```
firstNumber = int(input("Enter the first number: "))
secondNumber = int(input("Enter the second number: "))

if secondNumber != 0:
    print(firstNumber / secondNumber)
else:
    print("This operation cannot be done.")

print("THE END.")
```

```
Enter the first number: 2
Enter the second number: 3
0.6666666666666666
THE END.
```

Cara penanganan kesalahan seperti kode diatas dapat digunakan, tetapi memiliki kelemahan karena kode program dapat menjadi sangat kompleks dan besar

Untuk menangani ekspesi, python memanfaatkan blok yang disebut dengan `try...except` , dengan bentuk umum:

try: kode ..... ***except TipeEksekusi***: penanganan kesalahan

Jika kita mempunyaki kode-kode "mencurigakan" yang mungkin dapat menampilkan eksepsi,kita perlu menyimpan kode tersebut pada blok **try**.

Ketika kode berjalan normal, kode pada bagian **except** tidak akan dieksekusi. Sebaliknya jika terjadi kesalahan, maka eksekusi kode di bagian **try** akan dihentikan, dan program akan mengeksekusi pada bagian **except**

In [66]:
```
firstNumber = int(input("Enter the first number: "))
secondNumber = int(input("Enter the second number: "))

try:
    print(firstNumber / secondNumber)
except:
    print("This operation cannot be done.")

print("THE END.")
```

```
Enter the first number: 2
Enter the second number: 0
This operation cannot be done.
THE END.
```

In [69]:
```
try:
    print("antri") #jalan --> 1
    x = 1 / "isna" # eror ->-> xxxx
    print("2")
except:
    print("Oh dear, something went wrong...")

print("3")
```

```
antri
2
3
```

Block `try ... except` dapat digunakan untuk menangani lebih dari satu eksepsi, dengan menggunkan bentuk umum:

try: : except exc1: : except exc2: : except: :

Satu atau beberapa statement yang terdapat dalam blok `try` dapat menimbulkan lebih dari satu tipe ekspesi.

In [72]:
```
try:
    x = int(input("Enter a number: ")) #0
    y = 1 / x #eror
    print(y)
except ZeroDivisionError:
    print("Nggak bisa dibagi 0 mas bro")
except ValueError:
    print("Harus angka ya..")
except:
    print("Ada yang salah...")

print("THE END.")
```

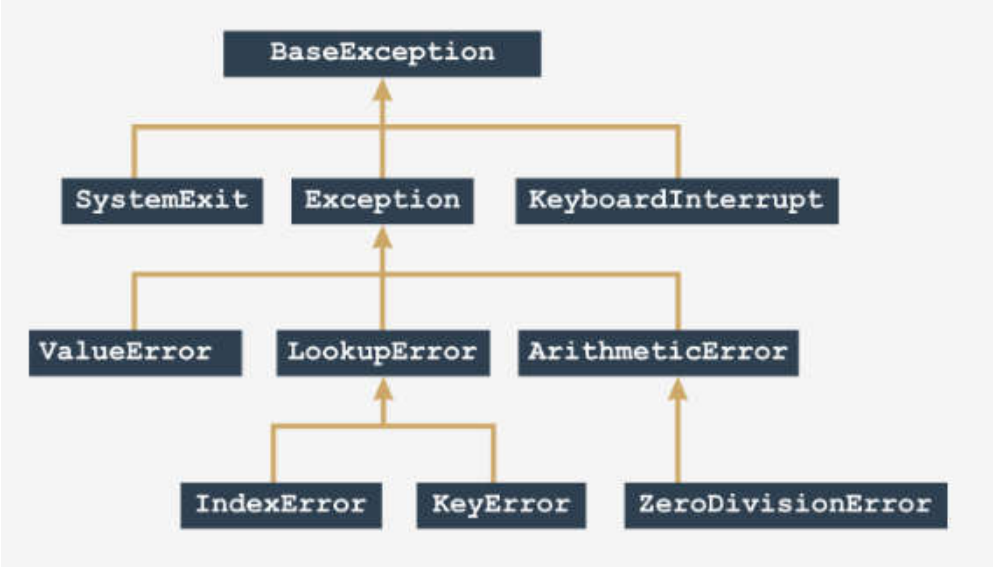
```
Enter a number: 0
Nggak bisa dibagi 0 mas bro
THE END.
```

Percobaan:

- Inputkan bilangan integer (e.g., 5)
- Inputkan 0
- Inputkan data non-integer

Built-in Exception

Python 3 menyediakan 63 built-in exceptions dan semuanya membentuk pohon hirarki



- ZeroDivisionError merupakan spesial kasus dari kelas eksepsi ArithmeticError;
- ArithmeticError merupakan spesial kasus dari kelas eksepsi Exception;
- Exception merupakan spesial kasus dari kelas eksepsi BaseException;

In [74]:
```
try:
    y = 1 / 0
except ArithmeticError:
    print("Oooppsss Ahaha...")
except ZeroDivisionError:
    print("Oooppsss...")

print("THE END.")

# Ubah ZeroDivisionError dengan ArithmeticError, Exception dan BaseException
```

```
Oooppsss Ahaha...
THE END.
```

Jika eksepsi muncul di dalam fungsi, maka eksepsi itu dapat ditangani dengan dua cara:

- di dalam fungsi
- di luar fungsi

In [75]:
```
def badFun(n):
    try:
        return 1 / n
    except ArithmeticError:
        print("Arithmetic Problem!")
    return None

badFun(0)

print("THE END.")
```

```
Arithmetic Problem!
THE END.
```

In [42]:
```
def badFun(n):
    return 1 / n

try:
    badFun(0)
except ArithmeticError:
    print("What happened? An exception was raised!")

print("THE END.")
```

```
What happened? An exception was raised!
THE END.
```

Raise

Eksepsi tertentu dapat kita panggil secara paksa dengan menggunakan perintah `raise` , meskipun sebenarnya ada kejadian yang menyebabkan jenis kesalahan tersebut

In [77]:
```
def badFun(n):
    raise ZeroDivisionError
```

`raise` digunakan untuk memanggil secara paksa dari suatu eksepsi, meskipun tidak ada kejadian yang menyebabkan jenis kesalahan tersebut.

In [6]:
```
def badFun(n):
    raise 1/0

try:
    badFun(0)
except ArithmeticError:
    print("What happened? An error?")

print("THE END.")
```

```
What happened? An error?
THE END.
```

In [80]:
```
def badFun(n):
    try:
        return n / 0
    except:
        print("Ada yang salah!")
        raise #muncul eror

try:
    badFun(1)

except NameError:
    print("I see name eror!")
except ArithmeticError:
    print("I see!")


print("THE END.")
```

```
Ada yang salah!
I see!
THE END.
```

Pemanggilan `raise` tanpa nama eksepsi hanya dapat dilakukan di dalam bagian `except`

Dari kode di atas `ZeroDivisionError` muncul sebanyak dua kali, yaitu:

- di dalam `try`
- di bagian `except` di dalam fungsi

Assert

```
assert expression
```

Fungsi assertion:

- Assert akan mengevaluasi ekspresi
- Jika ekspresi bernilai True atau nilai numerik bukan nol, atau string tidak kosong, atau nilai lain yang berbeda dari `None` tidak akan di eksekusi
- Jika selain itu akan muncul eksepsi `AssertionError`

Penggunaan assertion:

- kita dapat menggunakan assertion jika kita ingin kode yang dibuat benar-benar aman dari data kita belum yakin kebenarannya
- mengamankan kode dari hasil yang tidak valid
- assertion merupakan pelengkap `exception`

In [19]:
```
import math

x = float(input("Enter a number: "))
assert x >= 0.0

x = math.sqrt(x)

print(x)
```

```
Enter a number: -9
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-19-49fad9c6e200> in <module>
      2
      3 x = float(input("Enter a number: "))
----> 4 assert x >= 0.0
      5
      6 x = math.sqrt(x)

AssertionError:
```

Strings

Computers store characters as numbers

Every character used by a computer corresponds to a unique number, and vice versa

Some of these characters are called whitespaces, while others are named control characters, because their purpose is to control input/output devices.

ASCII (short for American Standard Code for Information Interchange) is the most widely used, and you can assume that nearly all modern devices (like computers, printers, mobile phones, tablets, etc.) use that code.

The code provides space for 256 different characters

I18N

The software I18N is a standard in present times. Each program has to be written in a way that enables it to be used all around the world, among different cultures, languages and alphabets.

Code points and code pages

A code point is a number which makes a character. For example, 32 is a code point which makes a space in ASCII encoding. We can say that standard ASCII code consists of 128 code points.

A code page is a standard for using the upper 128 code points to store specific national characters.

For example, the code point 200 makes Č (a letter used by some Slavic languages) when utilized by the ISO/IEC 8859-2 code page, and makes Ш (a Cyrillic letter) when used by the ISO/IEC 8859-5 code page.

Unicode

Code pages helped the computer industry to solve I18N issues for some time, but it soon turned out that they would not be a permanent solution.

Unicode assigns unique (unambiguous) characters (letters, hyphens, ideograms, etc.) to more than a million code points. The first 128 Unicode code points are identical to ASCII, and the first 256 Unicode code points are identical to the ISO/IEC 8859-1 code page (a code page designed for western European languages).

The Nature of Strings in Python

Python's strings are immutable sequences.

The `len()` function used for strings returns a number of characters contained by the arguments.

In [20]:
```
# Example 1

word = 'by'
print(len(word))


# Example 2

empty = ''
print(len(empty))


# Example 3

i_am = 'I\'m'
print(len(i_am))
```

```
2
0
3
```

`Multiline strings`

The string starts with three apostrophes, not one. The same tripled apostrophe is used to terminate it.

In [21]:
```
multiLine = '''Line #1
hjsksdalsdkjaljald
Line #2'''

print(len(multiLine))

# The missing character is simply invisible - it's a whitespace.
```

```
36
```

In [ ]:
```
multiLine = """Line #1
Line #2"""

print(len(multiLine))
```

Operations on strings

In general, strings can be:

- concatenated (joined) (+) The + operator used against two or more strings produces a new string containing all the characters from its arguments
- replicated. (*) The * operator needs a string and a number as arguments; in this case, the order doesn't matter - you can put the number before the string, or vice versa, the result will be the same

In [26]:
```
str1 = 'a'
str2 = 'b'

print(str1 + str2)
print(str2 + str1)
print(5 * 'ai')
print('bu ' * 4)
```

```
ab
ba
aiaiaiaiai
bu bu bu bu
```

github.com/isnaalfi >>> DigitalTalent

Operations on strings: ord()

If you want to know a specific character's ASCII/UNICODE code point value, you can use a function named ord() (as in ordinal).

In [27]:
```
# Demonstrating the ord() function

ch1 = 'a'
ch2 = ' ' # space

print(ord(ch1))
print(ord(ch2))
```

```
97
32
```

Operations on strings: chr()

The function takes a code point and returns its character.

In [ ]:
```
# Demonstrating the chr() function

print(chr(97))
print(chr(65))
```

In [ ]:
```
chr(ord('x')) == 'x'
ord(chr(x)) == x
```

Strings as sequences: indexing

Python's strings are sequences.

Strings aren't lists, but **you can treat them like lists in many particular cases**.

In [5]:
```
# If you want to access any of a string's characters, you can do it using indexi
ng,
# Indexing strings

exampleString = 'silly walks'

for ix in range(len(exampleString)):
    print(exampleString[ix], end='-')

print()
```

```
s-i-l-l-y- -w-a-l-k-s-
```

Strings as sequences: iterating

Iterating through the strings works, too.

In [6]:
```
# Iterating through a string

exampleString = 'silly walks'

for ch in exampleString:
    print(ch, end=' ')

print()
```

```
s i l l y   w a l k s
```

Slices

In [7]:
```
# Slices

alpha = "abdefg"

print(alpha[1:3])
print(alpha[3:])
print(alpha[:3])
print(alpha[3:-2])
print(alpha[-3:4])
print(alpha[::2]) # ini artinya cari mulai depan--> akhir, per 2 chr
print(alpha[1::2]) # ini artinya cari mulai 1--> akhir, per 2 chr
```

```
bd
efg
abd
e
e
adf
beg
```

The `in` and `not in` operators

The `in` operator checks if its left argument (a string) can be found anywhere within the right argument (another string).

The result of the check is simply `True` or `False`

In [ ]:
```
alphabet = "abcdefghijklmnopqrstuvwxyz"

print("f" in alphabet)
print("F" in alphabet)
print("1" in alphabet)
print("ghi" in alphabet)
print("Xyz" in alphabet)
```

In [8]:
```
alphabet = "abcdefghijklmnopqrstuvwxyz"

print("f" not in alphabet)
print("F" not in alphabet)
print("1" not in alphabet)
print("ghi" not in alphabet)
print("Xyz" not in alphabet)
```

```
False
True
True
False
True
```

Python strings are immutable

- It doesn't allow you to use the `del` instruction to remove anything from a string.
- You can do with del and a string is to remove the string as a whole.
- Python strings don't have the `append()` method
- The `insert()` method is **illegal**, too

In [ ]:
```
alphabet = "abcdefghijklmnopqrstuvwxyz"

del alphabet
print(alphabet)
```

In [ ]:
```
alphabet = "abcdefghijklmnopqrstuvwxyz"

alphabet.append("A")
```

In [ ]:
```
alphabet = "abcdefghijklmnopqrstuvwxyz"

alphabet.insert(0, "A")
```

In [15]:
```
alphabet = "bcdefghijklmnopqrstuvwxy"

alphabet = "isna"+" "+alphabet
alphabet = alphabet + "z"

print(alphabet)
```

```
isna bcdefghijklmnopqrstuvwxyz
```

Operations on strings: min()

The function finds the **minimum element** of the sequence passed as an argument. **minimum disini minimum nilai ASCII nya**

There is one condition - the sequence (string, list, it doesn't matter) cannot be empty

In [16]:
```
# Demonstrating min() - Example 1
print(min("aAbByYzZ"))


# Demonstrating min() - Examples 2 & 3
t = 'The Knights Who Say "Ni!"'
print('[' + min(t) + ']')

t = [0, 1, 2]
print(min(t))
```

```
A
[ ]
0
```

Operations on strings: max()

A function named `max()` finds the maximum element of the sequence.

In [ ]:
```
# Demonstrating max() - Example 1
print(max("aAbByYzZ"))


# Demonstrating max() - Examples 2 & 3
t = 'The Knights Who Say "Ni!"'
print('[' + max(t) + ']')

t = [0, 1, 2]
print(max(t))
```

Operations on strings: the `index()` method

The `index()` method (it's a method, not a function) searches the sequence from the beginning, in order to find the first element of the value specified in its argument.

The element searched for must occur in the sequence - its absence will cause a ValueError exception.

The method returns the index of the first occurrence of the argument which means that the lowest possible result is 0, while the highest is the length of argument decremented by 1.

In [17]:
```
# Demonstrating the index() method
print("aAbByYzZaA".index("b"))
print("aAbByYzZaA".index("Z"))
print("aAbByYzZaA".index("A"))
```

```
2
7
1
```

Operations on strings: the list() function

The list() function takes its argument (a string) and creates a new list containing all the string's characters, one per list element.

`list()` is able to create a new list from many other entities (e.g., from tuples and dictionaries).

Operations on strings: the count() method

The count() method counts all occurrences of the element inside the sequence.

The absence of such elements doesn't cause any problems.

In [18]:
```
# Demonstrating the list() function
print(list("abcabc"))

# Demonstrating the count() method
print("abcabc".count("b"))
print('abcabc'.count("d"))
```

```
['a', 'b', 'c', 'a', 'b', 'c']
2
0
```

String Method

The capitalize() method

The capitalize() method creates a new string filled with characters taken from the source string, but it tries to modify them in the following way:

- if the first character inside the string is a letter, it will be converted to upper-case
- all remaining letters from the string will be converted to lower-case.

In [19]:
```
print("Alpha".capitalize())
print("Alpha".upper()) # ALPHA
print('ALPHA'.capitalize())
print(' Alpha'.capitalize())
print('123'.capitalize())
print("αβγδ".capitalize())
```

```
Alpha
ALPHA
Alpha
 alpha
123
Αβγδ
```

The center() method

The `center()` method makes a copy of the original string, trying to center it inside a field of a specified width

The centering is actually done by adding some spaces before and after the string.

In [29]:
```
print('[' + 'Beta'.center(2) + ']')
print('[' + 'Beta'.center(7,"!") + ']')
print('[' + 'Beta'.center(6) + ']')
```

```
[Beta]
[!!Beta!]
[ Beta ]
```

The two-parameter variant of center() makes use of the character from the second argument, instead of a space.

In [ ]:
```
print('[' + 'gamma'.center(20, '*') + ']')
```

In [30]:
```
print("*".center(10))
print("* *".center(10))
print("*   *".center(10))
print("*     *".center(10))
print("***   ***")
print("*   *".center(10))
print("*   *".center(10))
print("*****".center(10))
```

```
    *
   * *
  *   *
 *     *
***   ***
  *   *
  *   *
  *****
```

The endswith() method

The `endswith()` method checks if the given string ends with the specified argument and returns `True` or `False`, depending on the check result.

In [ ]:
```
t = "zeta"
print(t.endswith("a"))
print(t.endswith("A"))
print(t.endswith("et"))
print(t.endswith("eta"))
```

In [31]:
```
# Demonstrating the endswith() method
if "epsilon".endswith(".com"):
    print("yes")
else:
    print("no")
```

```
no
```

In [ ]:
```
teksnya=input("Masukan teks anda ")

if teksnya.endswith(".com"):
    print("Nama website")
else:
    print("Oh bukan website, kisanak")
```

The find() method

The find() method is similar to index(), it looks for a substring and returns the index of first occurrence of this substring, but:

it doesn't generate an error for an argument containing a non-existent substring

it works with strings only

In [ ]:
```
t = 'theta'
print(t.find('eta'))
print(t.find('et'))
print(t.find('the'))
print(t.find('ha'))
```

In [37]:
```
print('kappa'.find('a', 2))

# proses pencarian dimulai dari indeks 2, proses pencarian akan berhenti jika su
dah ditemukan huruf 'a'
```

```
4
```

In [ ]:
```
txt = """A variation of the ordinary lorem ipsum
text has been used in typesetting since the 1960s
or earlier, when it was popularized by advertisements
for Letraset transfer sheets. It was introduced to
the Information Age in the mid-1980s by the Aldus Corporation,
which employed it in graphics and word-processing templates
for its desktop publishing program PageMaker (from Wikipedia)"""

fnd = txt.find('the')
while fnd != -1:
    print(fnd)
    fnd = txt.find('the', fnd + 1)
```

In [38]:
```
print('kappa'.find('a', 1, 4))
print('kappa'.find('a', 2, 4))

# argumen ketiga menunjuk ke indeks pertama yang tidak akan dipertimbangkan sela
ma pencarian
```

```
1
-1
```

In [ ]:
```
# Demonstrating the find() method
print("Eta".find("ta"))
print("Eta".find("mma"))
```

The isalnum() method

The parameterless method named isalnum() checks if the string contains only digits or alphabetical characters (letters), and returns True or False according to the result.

In [ ]:
```
# Demonstrating the isalnum() method
print('lambda30'.isalnum())
print('lambda'.isalnum())
print('30'.isalnum())
print('@'.isalnum())
print('lambda_30'.isalnum())
print(''.isalnum())
```

The isalpha() method

The isalpha() method is more specialized - it's interested in letters only.

The isdigit() method

In turn, the isdigit() method looks at digits only

In [ ]:
```
# Example 1: Demonstrating the isapha() method
print("Moooo".isalpha())
print('Mu40'.isalpha())

# Example 2: Demonstrating the isdigit() method
print('2018'.isdigit())
print("Year2019".isdigit())
```

The islower() method

The islower() method is a fussy variant of isalpha()

It accepts lower-case letters only.

The isspace() method

The isspace() method identifies whitespaces only

The isupper() method

The isupper() method is the upper-case version of islower()

It concentrates on upper-case letters only.

In [ ]:
```
# Example 1: Demonstrating the islower() method
print("Moooo".islower())
print('moooo'.islower())
print("")
# Example 2: Demonstrating the isspace() method
print(' \n '.isspace())
print(" ".isspace())
print("mooo mooo mooo".isspace())
print("")
# Example 3: Demonstrating the isupper() method
print("Moooo".isupper())
print('moooo'.isupper())
print('MOOOO'.isupper())
```

The join() method

In [ ]:
```
# Demonstrating the join() method
print("*".join(["omicron", "pi", "rho"]))

# the join() method is invoked from within a string containing a comma
# the join's argument is a list containing three strings;
# the method returns a new string.
```

The lower() method

The lower() method makes a copy of a source string, replaces all upper-case letters with their lower-case counterparts, and returns the string as the result.

In [ ]:
```
# Demonstrating the lower() method
print("SiGmA=60".lower())
```

The lstrip() method

The parameterless lstrip() method returns a newly created string formed from the original one by removing all leading whitespaces.

The one-parameter lstrip() method, removes all characters enlisted in its argument

In [ ]:
```
# Demonstrating the lstrip() method
print("[" + "       tau ".lstrip() + "]")
```

In [ ]:
```
print("["+" ilang spasi nih ".lstrip())
```

In [ ]:
```
print("www.ugm.w".rstrip(".w"))
print("pythoninstitute.org".lstrip("thpy"))
```

The replace() method

The two-parameter replace() method returns a copy of the original string in which all occurrences of the first argument have been replaced by the second argument

The three-parameter replace() variant uses the third argument (a number) to limit the number of replacements.

In [39]:
```
# Demonstrating the replace() method
print("www.netacad.com".replace("netacad.com", "pythoninstitute.org"))
print("This is it!".replace("is", "are"))
print("Apple juice".replace("juice", ""))
```

```
www.pythoninstitute.org
Thare are it!
Apple
```

In [42]:
```
print("Mail jual ayam dua seringgit, dua seringgit!. Walau kadang bayarnya cuma
seringgit".replace("seringgit", "sepuluh ribu"))
print("Ipin kalau betul suka bilang betul betul betul!".replace("betul", "bena
r", 1)) # 1 di arrgument ke-3 adalah maksimum yg di replace
print("This is it!".replace("is", "are", 2))
```

```
Mail jual ayam dua sepuluh ribu, dua sepuluh ribu!. Walau kadang bayarnya cuma
sepuluh ribu
Ipin kalau benar suka bilang betul betul betul!
Thare are it!
```

The rfind() method

Start their searches from the end of the string

hence the prefix `r` , for `right`

In [43]:
```
# Demonstrating the rfind() method
print("tau tau tau".rfind("ta"))
print("tau tau tau".rfind("ta", 9))
print("tau tau tau".rfind("ta", 3, 9))
```

```
8
-1
4
```

In [ ]:
```
# Demonstrating the rfind() method
print("0123456789sepuluh".rfind("9"))
print("0123456789sepuluh".rfind("9", 9)) #dimulai dari index ke-9
print("0123456789sepuluh".rfind("9", 3, 9))#dimulai dari index ke 3 -> (sebelum)
9, -1 artinya gak nemu
```

The rstrip() method

Two variants of the rstrip() method do nearly the same as lstrips, but affect the opposite side of the string.

In [ ]:
```
# Demonstrating the rstrip() method
print("[" + " upsilon ".rstrip() + "]")
print("cisco.com".rstrip(".com"))
```

The split() method

The split() method does what it says - it splits the string and builds a list of all detected substrings.

The method assumes that the substrings are delimited by whitespaces

In [ ]:
```
# Demonstrating the split() method
print("phi       chi\npsi".split())
```

The startswith() method

The startswith() method is a mirror reflection of endswith() - it checks if a given string starts with the specified substring.

In [ ]:
```
# Demonstrating the startswith() method
print("omega".startswith("meg"))
print("omega".startswith("om"))
```

The strip() method

The trip() method combines the effects caused by rstrip() and lstrip() - it makes a new string lacking all the leading and trailing whitespaces.

In [ ]:
```
# Demonstrating the strip() method
print("[" + "   aleph   ".strip() + "]")
```

The swapcase() method

The swapcase() method makes a new string by swapping the case of all letters within the source string: lower-case characters become upper-case, and vice versa.

The title() method

It changes every word's first letter to upper-case, turning all other ones to lower-case.

The upper() method

The upper() method makes a copy of the source string, replaces all lower-case letters with their upper-case counterparts, and returns the string as the result.

In [ ]:
```
# Demonstrating the swapcase() method
print("ini TULISAN alay.".swapcase())
print()
# Demonstrating the title() method
print("ini TULISAN alay.. Part 1.".title())
print()
# Demonstrating the upper() method
print("ini TULISAN alay.. Part 2.".upper())
```

LAB 5.1.9.18

In [ ]:
```
def mysplit(strng):
    # return [] if string is empty or contains whitespaces only
    if strng == '' or strng.isspace():
        return [ ]
    # prepare a list to return
    lst = []
    # prepare a word to build subsequent words
    word = ''
    # check if we are currently inside a word (i.e., if the string starts with a
word)
    inword = not strng[0].isspace()
    # iterate through all the characters in string
    for x in strng:
        # if we are currently inside a string...
        if inword:
            # ... and current character is not a space...
            if not x.isspace():
                # ... update current word
                word = word + x
            else:
                # ... otherwise, we reached the end of the word so we need to ap
pend it to the list...
                lst.append(word)
                # ... and signal a fact that we are outside the word now
                inword = False
        else:
            # if we are outside the word and we reached a non-white character...
            if not x.isspace():
                # ... it means that a new word has begun so we need to remember
it and...
                inword = True
                # ... store the first letter of the new word
                word = x
            else:
                pass
    # if we left the string and there is a non-empty string in word, we need to
update the list
    if inword:
        lst.append(word)
    # return the list to invoker
    return lst

print(mysplit("To be or not to be, that is the question"))
print(mysplit("To be or not to be,that is the question"))
print(mysplit("   "))
print(mysplit(" abc "))
print(mysplit(""))
```

String in Action

Comparing strings

Python's strings can be compared using the same set of operators which are in use in relation to numbers.

It just compares code point values, character by character.

- ==
- !=
- >
- >=
- <
- <=

In [ ]:
```
print('alpha' == 'alpha')
print ('alpha' != 'Alpha')
print ('alpha' < 'alphabet')
# String comparison is always case-sensitive (upper-case letters are taken as le
sser than lower-case).
print ('beta' > 'Beta')
print()
# Even if a string contains digits only, it's still not a number.
print('10' == '010')
print('10' > '010')
print('10' > '8')
print('20' < '8') #ini true looh
print('20' < '80')
```

In [ ]:
```
# Apakah output dari kode program di bawah ini
print('10' == 10)
print('10' != 10)
print('10' == 1)
print('10' != 1)
print('10' > 10)
```

Sorting

- The first is implemented as a function named sorted().
  - The function takes one argument (a list) and returns a new list
- The second function named `sort()`
  - The second method affects the list itself - no new list is created

In [ ]:
```
# Demonstrating the sorted() function
firstGreek = ['omega', 'alpha', 'pi', 'gamma']
firstGreek2 = sorted(firstGreek)

print(firstGreek)
print(firstGreek2)

print()

# Demonstrating the sort() method
secondGreek = ['omega', 'alpha', 'pi', 'gamma']
print(secondGreek)

secondGreek.sort(reverse=True)
print(secondGreek)
```

In [ ]:
```
# Demonstrating the sorted() function
firstGreek = "Alamak"
firstGreek2 = sorted(firstGreek)

print(firstGreek)
print(firstGreek2) #ini hasil sorted, dalam bentuk List

#coba join hasilnya
print("".join(firstGreek2)) # ini join hasil sorted
print(firstGreek[::-1])  #ini untuk reserve (mbalik teks)

print()

# Demonstrating the sort() method
secondGreek = ["a","A"]
print(secondGreek)

secondGreek.sort()
print(secondGreek)
```

Strings vs. numbers

How to convert a number (an integer or a float) into a string, and vice versa.

In [ ]:
```
itg = 13
flt = 1.3
si = str(itg)
sf = str(flt)
#print(type(si))
print(si + ' ' + sf)
```

LAB 5.1.10.6

In [ ]:
```
digits = [ '1111110',    # 0    ini untuk mempermudak, bayangkan ini nomor LED
           '0110000',    # 1
           '1101101',    # 2
           '1111001',    # 3
           '0110011',    # 4
           '1011011',    # 5
           '1011111',    # 6
           '1110000',    # 7
           '1111111',    # 8
           '1111011',    # 9
           ]

def printNumber(num):
    global digits #berlaku global, diambil dari digit atas
    #print(digits)
    digs = str(num)  # KENAPA dibuat str? biar kita bisa tahu jumlah digitnya..
    lines = [ '' for l in range(5) ] # ini untuk membuat tempat lampu (ada 5 bar
is)
    for d in digs:
        segs = [ [' ',' ',' '] for l in range(5) ] # membuat baris lampunya -->
initinya mau bikin matrix 5x3
        ptrn = digits[ord(d) - ord('0')] # ini mencari elem dalam list berdasark
an selisih nilai ascii nya. ex: 49-48 = 1
        #ptrn = singkatan patern/ pola lampu
        #print(ptrn)
        if ptrn[0] == '1':
            segs[0][0] = segs[0][1] = segs[0][2] = '#'
        if ptrn[1] == '1':
            segs[0][2] = segs[1][2] = segs[2][2] = '#'
        if ptrn[2] == '1':
            segs[2][2] = segs[3][2] = segs[4][2] = '#'
        if ptrn[3] == '1':
            segs[4][0] = segs[4][1] = segs[4][2] = '#'
        if ptrn[4] == '1':
            segs[2][0] = segs[3][0] = segs[4][0] = '#'
        if ptrn[5] == '1':
            segs[0][0] = segs[1][0] = segs[2][0] = '#'
        if ptrn[6] == '1':
            segs[2][0] = segs[2][1] = segs[2][2] = '#'
        for l in range(5):
            lines[l] += ''.join(segs[l]) + ' '
    for l in lines:
        print(l)

printNumber(int(input("Enter the number you wish to display: ")))
```

In [ ]:
```
# Simple program

# Caesar cipher
text = input("Enter your message: ")
cipher = ''
for char in text:
    if not char.isalpha():  #cek kalau bukan alphabet maka kontinue
        continue
    char = char.upper()  #membuat uppercase
    code = ord(char) + 1  # mencari nilai ASCII nya +1 biar geser
    if code > ord('Z'):  # ini case kalau Z balik ke A
        code = ord('A')
    cipher += chr(code) #ini untuk balikin dari code ascii ke character

print(cipher)
```

Palindrom

In [ ]:
```
text = input("Enter text: ")

# menghapus spasi/ whitespace
text = text.replace(' ','')

# ngecek kalau katanya sama pas dibalik/reverse
#[::-1] artinya, bikin slice dari start--> end, dengan langkah mundur (-1)
#kasus misal : isnansi ada 8 karakter maka [::-1] artinya sama dnegan  [8:0:-1]
#pengambilan slice dimulai dari indeks ke 7 (sebelum 8), sampai ke 0, dengan cat
atan langkah dari char paling belakang, per -1
if len(text) > 1 and text.upper() == text[::-1].upper():
    print("It's a palindrome")
else:
    print("It's not a palindrome")
```

In [ ]:
```

```