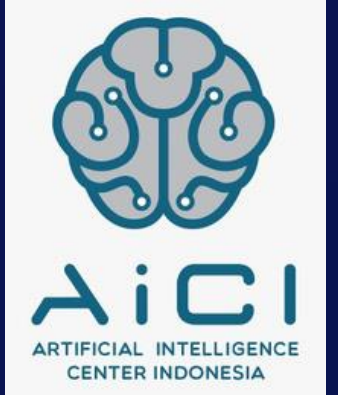




**Kampus
Merdeka**
INDONESIA JAYA



Exception Handling, Modules and Packages



Penyusun Modul: Fitria Yunita Dewi

Editor: Delyanda Rahmadini



Exception Handling

- Program Python akan berhenti jika saat dijalankan menemukan *error*
- Terdapat 2 macam *error* dalam Python:
 - *Syntax Error (Design-Time error)*
 - *Exception (Run-Time error)*
- *Exception Handling*: Proses merespon munculnya *exception* selama program dijalankan





Syntax Error

Syntax Error berkaitan dengan kesalahan pada penulisan sintaks pemrograman. Contohnya:

```
[1]: 1 # Ex:
      2 # SyntaxError
      3 print ( 'Python Parser error' ))
```

```
[1]: SyntaxError: unmatched ')'
```

```
[2]: 1 # Ex:
      2 # SyntaxError: EOL while scanning string literal
      3 a = "12"
      4 print (a)
```

```
[2]: SyntaxError: EOL while scanning string literal
```

```
[3]: 1 # Ex:
      2 # IndentationError: expected an indented block
      3 def myFunc ():
      4     # function scope
      5 print ( 'A' )
```

```
[3]: IndentationError: expected an indented block
```




Exception

Jika penulisan program sudah benar dan tidak ada sintaks yang salah, maka error yang muncul ketika dijalankan (*run-time*) disebut sebagai *exception*
Contoh:

```
2 # ZeroDivisionError: division by zero
3 a = 7 / 0
```

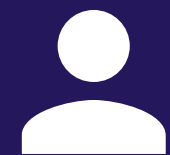
```
[4]: ZeroDivisionError: division by zero
```

```
[5]: 1 # Ex:
      2 # NameError: name 't' is not defined
      3 print (t)
```

```
[5]: NameError: name 't' is not defined
```

```
[6]: 1 # Ex:
      2 a = 12
      3 b = 'B'
      4 # TypeError: unsupported operand type(s) for +: 'int'
      5    and 'str'
      6 print (a + b)
```

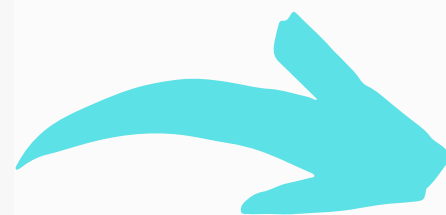
```
[6]: TypeError: unsupported operand type(s) for +: 'int' and
      'str'
```



Try-Except

- Seberusaha apapun kita membuat program bersih dari '*bug*' pada akhirnya kemungkinan besar tetap akan menemukan *error*
- Untuk membuat program tetap berjalan baik meskipun bertemu *error* (tidak terjadi *crashing*) dapat menggunakan blok `try-except`
- Jika ditemukan *error* dalam blok `try`, maka eksekusi kode akan berpindah langsung menuju blok `except`
- Setiap baris kode di bawah lokasi terjadi *error* dalam blok `try` tidak akan dijalankan
- Jika tidak terdapat *error* maka blok `except` akan diskip

```
try :
    ....
    .....
    ..... error .....
    ..... will not be executed
    ..... will not be executed
    ..... will not be executed
except :
    .....
    .....
```



Pseudocode blok `try-except`



Multiple Except Blocks

```
try :  
    ....  
    ....  
    error  
    ....  
except Exception_Type_1:  
    ... actions for type 1. ...  
    ....  
except Exception_Type_2:  
    ... actions for type 2. ...  
    ....
```

Pseudocode multiple blok
except

Contoh dalam operasi pembagian matematika, kemungkinan *error* yang muncul adalah:

- Salah satu dari dua input bukan merupakan nilai numerik/angka
- Input pembagi merupakan angka namun bernilai 0

Pada kasus ini dapat digunakan 2 buah blok **except**

```
except ValueError:  
    print('The inputs are not numeric...')  
  
except ZeroDivisionError:  
    print('Second number cannot be ZERO....')
```

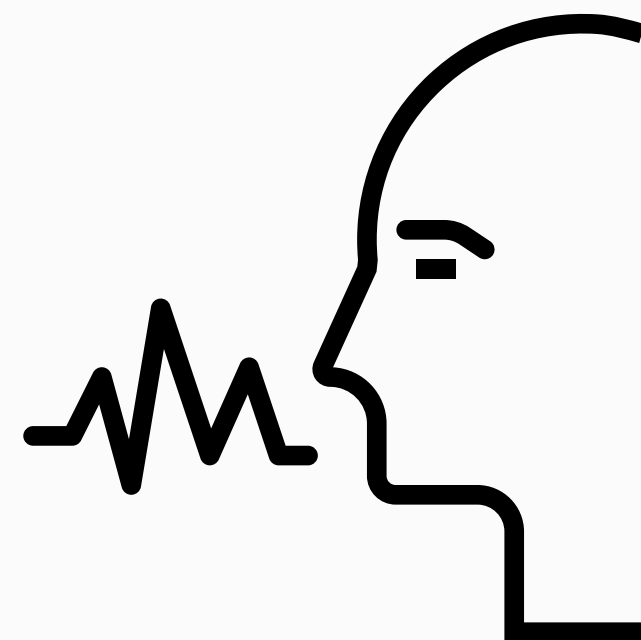



Else

Blok **try-except** memiliki klausa opsional **else** yang akan dijalankan jika seluruh block **except** tidak dijalankan



```
try :  
    <---- code ---->  
except :  
    <---- error ---->  
else :  
    <---- no errors ---->
```





Latihan Exception Handling

- Buat sebuah program yang meminta pengguna memasukkan input
- Jika inputnya merupakan bilangan, ubah dalam bentuk integer, dan munculkan nilai kuadratnya
- Jika inputnya berupa string, munculkan kembali dalam huruf kapital
- Gunakan **try-except** !

Solusi:





Latihan Exception Handling

- Buat sebuah program yang meminta pengguna memasukkan input
- Jika inputnya merupakan bilangan, ubah dalam bentuk integer, dan munculkan nilai kuadratnya
- Jika inputnya berupa string, munculkan kembali dalam huruf kapital
- Gunakan **try-except** !

Solusi:

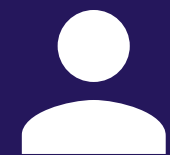
```
try:
    user_input = input("Silakan memasukkan suatu nilai: ")
    nilai = int(user_input)**2
except:
    nilai = user_input.upper()

print(nilai)
```



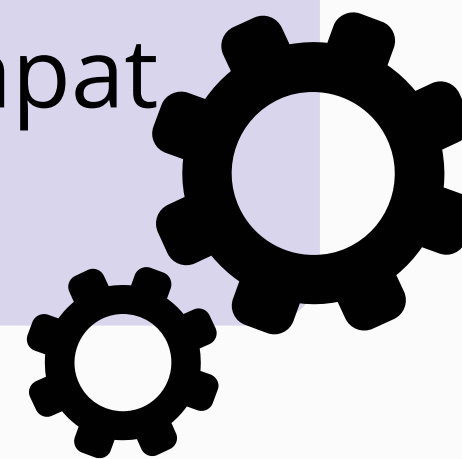
Modules & Packages





Modules

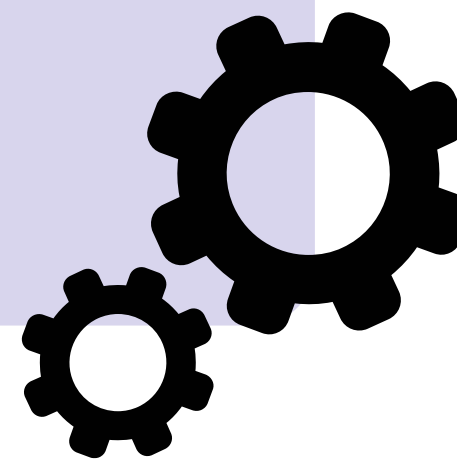
- Python termasuk bahasa pemrograman yang modular
- Manfaat pemrograman modular:
 - Mencegah pembuatan kode yang terlalu berulang, cukup mendefinisikan suatu modul kemudian dapat digunakan kapan pun dibutuhkan
 - Organisasi file menjadi lebih rapi
 - *Code maintenance* menjadi lebih mudah
- *Function* berisi blok kode yang spesifik sedangkan *module* dapat berisi banyak *function*





Standard/Built-in Module

- os module.
- random module.
- math module.
- time module.
- sys module.
- collections module.
- statistics module.





Defining Custom Modules

- Apapun *script* python yang kita buat jika berisi 1 atau lebih fungsi *custom* dapat menjadi sebuah *module*.
- Contoh suatu file bernama `bilangan.py` berisikan empat buah fungsi sebagai berikut:

```
def bulat():  
    print("Seluruh bilangan yang terdiri dari bilangan  
negatif, nol dan positif")  
  
def asli():  
    print("Seluruh bilangan bulat yang positif")  
  
def cacah():  
    print("Seluruh bilangan bulat yang lebih dari atau sama  
dengan 0")  
  
def isGenap(num):  
    try:  
        if float(num)%2 == 0:  
            print("{0} adalah bilangan genap".format(num))  
        else:  
            print("{0} bukan bilangan genap".format(num))  
    except:  
        print("{0} bukan termasuk bilangan".format(num))
```



Defining Custom Modules

Semua fungsi pada `bilangan.py` dapat diakses di luar file tersebut dengan cara mengimport nama dari file tersebut dalam direktori yang sama

```
import bilangan

bilangan.bulat()
bilangan.asli()
bilangan.cacah()
bilangan.isGenap(7)
bilangan.isGenap("4")
bilangan.isGenap("xxx")
```

Seluruh bilangan yang terdiri dari bilangan negatif, nol dan positif

Seluruh bilangan bulat yang positif

Seluruh bilangan bulat yang lebih dari atau sama dengan 0

7 bukan bilangan genap

4 adalah bilangan genap

xxx bukan termasuk bilangan



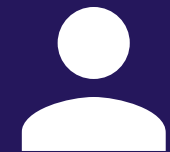
How Python Find Modules?

- Mencari dalam folder yang sama dengan file *script* yang sedang dijalankan
- Mencari dalam *variable environment* PYTHONPATH. Untuk *custom module*, direktorinya perlu ditambahkan secara manual
- Mencari dalam folder dimana Python diinstall

Untuk mengetahui dimana saja direktori yang dicari oleh Python









```
print('Python Search Paths for Modules:')  
for path in sys.path:  
    print(path)  
# sys.path -> the list where Python looks for modules
```



System Level Access

Untuk membuat *module* yang sudah dibuat dapat diakses secara global dari project manapun (tidak harus dalam direktori yang sama), file *module* dapat dipindahkan ke direktori di mana Python terinstall

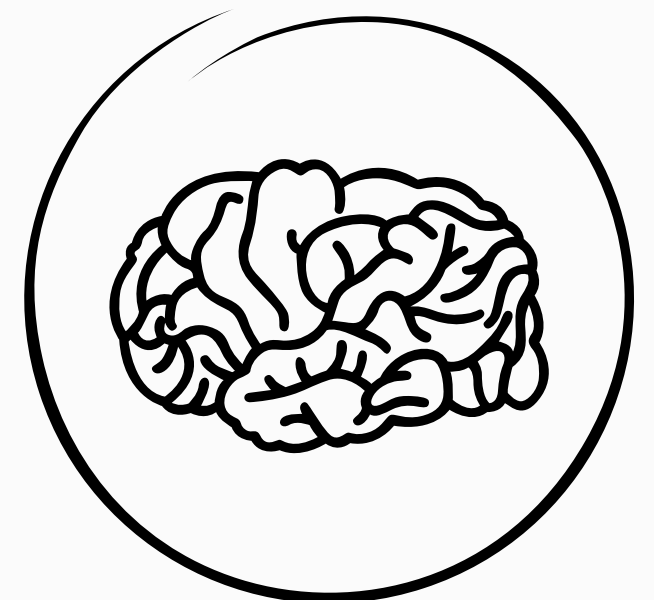
> AiCIB > AppData > Local > Programs > Python > Python310 > Lib >			
<input type="checkbox"/> Name	Type	Size	
 input_operations	Python Source File	1 KB	
 first_module	Python Source File	1 KB	
 imp	Python Source File	11 KB	
 inspect	Python Source File	125 KB	
 io	Python Source File	5 KB	
 ipaddress	Python Source File	76 KB	





Packages

- Python *package* pada hakikatnya adalah folder
- Python *package* biasanya terdiri dari beberapa *module*
- Perbedaan *package* dengan folder biasa adalah, di dalamnya harus terdapat file `__init__.py`





Create a Package

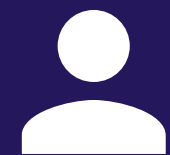
Contoh

Buat suatu package bernama `pack` yang terdiri dari 2 modul:

Masing-masing modul berisi 1 buah fungsi

```
mod_1.py X
1
2  def print_mod_1():
3  |    print('Module 1')
```

```
mod_2.py X
1
2  def print_mod_2():
3  |    print('Module 2')
```



Create a Package

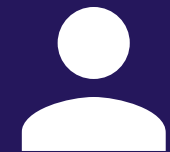
File `__init__.py` dibutuhkan agar Python dapat melihat suatu folder sebagai sebuah *package*

Hal yang dibutuhkan ada di dalam `__init__.py`:

- *Sub-package* (jika ada) dan modul-modul yang perlu diimport
- Variabel global
- *documentation*

File yang ada di dalam folder `pack`

```
▼ pack
  ├── __init__.py
  ├── mod_1.py
  └── mod_2.py
```



Inside of `__init__.py`

`__init__.py` X

```
1  """
2  This package has two modules:
3
4  * mod_1
5  * mod_2
6
7  """
8  # wrong way using absolute import
9  # import mod_1
10 # import mod_2
11
12 # correct import using relative import
13 from . import mod_1
14 from . import mod_2
15
```

Pada contoh berikut tidak ada variable global

Documentation, dapat diakses menggunakan `__doc__`

Jangan gunakan *absolute import* karena akan muncul error **ModuleNotFoundError**

Import seluruh *module* pada package (`mod_1`, `mod_2`) menggunakan *relative import*



Import the Package

Mengakses *documentation*

```
import pack  
  
print(pack.__doc__)
```

This package has two modules:

```
* mod_1  
* mod_2
```

Mengakses fungsi dan modul di dalam package

Cara 1:

```
import pack  
  
pack.mod_1.print_mod_1()  
pack.mod_2.print_mod_2()
```

Module 1

Module 2

Cara 2:

```
from pack import mod_1, mod_2  
  
mod_1.print_mod_1()  
mod_2.print_mod_2()
```

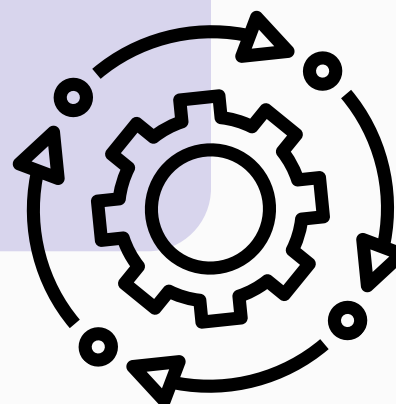
Module 1

Module 2



Installing Packages

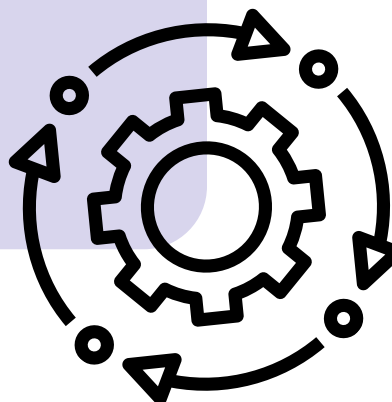
- Banyak cara untuk menginstal *third-party package*, cara yang paling umum dengan menggunakan *package manager* **pip**
- **Pip** sendiri merupakan sebuah *package*, pip sudah terinstal otomatis pada saat Python diinstal
- **PyPI** (Python Package Index) adalah repository *software official* untuk bahasa pemrograman Python. PyPI dapat membantuk kita untuk menginstal *software* yang sudah dikembangkan dan dipublish dalam komunitas Python





Most Popular Python Packages

- Numpy
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- SciPy
- Pytorch
- Requests
- Urllib3
- NLTK
- Etc.





Latihan Module & Package

- Buatlah modul bernama `vowels` yang berisi fungsi bernama `get_vowels(text)` dengan sebuah parameter input teks. Fungsi akan menghasilkan output berupa kumpulan huruf vocal yang ada pada teks input (hint: gunakan tipe data *set* untuk kumpulan huruf)
- Buatlah modul kedua dengan prinsip yang sama namun bernama `consonants`, berisi fungsi `get_consonants(text)` dan output berupa kumpulan huruf konsonan
- Kumpulkan kedua modul ke dalam 1 *package* bernama `solution`





Solusi:

Latihan Module & Package

```
__init__.py x
1 """
2 Solusi Latihan Module & Package
3 """
4
5 # import all modules
6 from . import consonants, vowels
7
```

```
vowels.py x
1 """
2 vowels module.
3 """
4
5 def get_vowels(text):
6
7     vowels_set = set()
8     vowels_list = 'aeiou'
9
10    for letter in text:
11        if letter.lower() in vowels_list and letter.isalpha():
12            vowels_set.add(letter)
13
14    return vowels_set
15
```

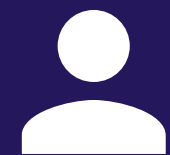
```
consonants.py x
2 consonants module.
3 """
4
5 def get_consonants(text):
6
7     consonants_set = set()
8     vowels_list = 'aeiou'
9
10    for letter in text:
11        if letter.lower() not in vowels_list and letter.isalpha():
12            consonants_set.add(letter)
13
14    return consonants_set
15
```

✓ solution

__init__.py

consonants.py

vowels.py



Solusi:

Latihan Module & Package

```
from solution import vowels, consonants

text = "AI Talents Program"

vows_in_text = vowels.get_vowels(text)
cons_in_text = consonants.get_consonants(text)

print("Huruf-huruf vokal di dalam 'AI Talents Program' adalah", vows_in_text)
print("Huruf-huruf konsonan di dalam 'AI Talents Program' adalah", cons_in_text)
```

Huruf-huruf vokal di dalam 'AI Talents Program' adalah {'A', 'e', 'a', 'o', 'I'}

Huruf-huruf konsonan di dalam 'AI Talents Program' adalah {'g', 'T', 't', 's', 'l', 'r', 'm', 'P', 'n'}