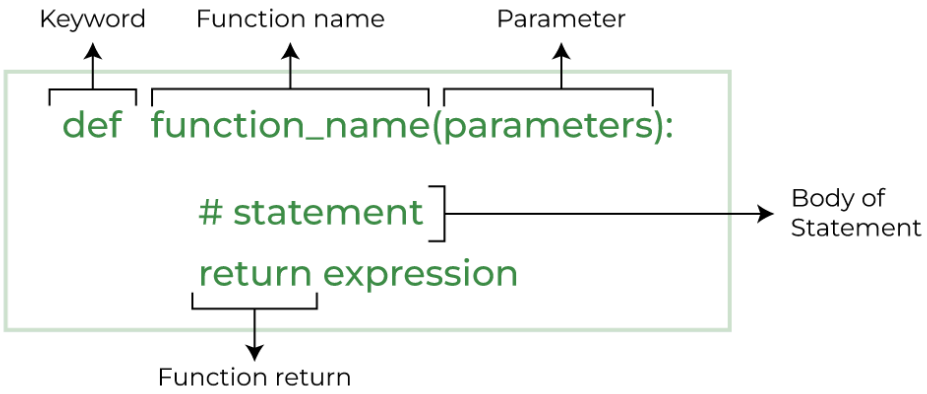
BAB V

Function, Error Handling & RegEx

A. Function

1. Pembuatan Function

Fungsi dalam pemrograman Python adalah sekumpulan perintah atau kode yang dikelompokkan menjadi satu kesatuan yang akan dieksekusi saat dipanggil. Fungsi dapat menerima parameter, mengembalikan nilai, dan dapat dipanggil berkali-kali secara independen. Dalam Python, terdapat dua jenis fungsi yang bisa digunakan. Fungsi bawaan adalah fungsi yang telah ada dalam bahasa Python dan bisa langsung digunakan, seperti len(), max(), print(), dan sorted(). Di sisi lain, kita bisa membuat fungsi sendiri. Fungsi ini dapat definisikan sesuai kebutuhan program dan memberikan fleksibilitas dalam merangkai serangkaian tugas yang lebih spesifik.



Dalam pemrograman Python, terdapat dua jenis fungsi berdasarkan bagaimana mereka berinteraksi dengan nilai. Pertama, **fungsi yang mengembalikan nilai** adalah fungsi yang memberikan keluaran berupa nilai setelah menjalankan serangkaian pernyataan di dalamnya. Penggunaan keyword `**return**` digunakan untuk mengirimkan nilai tersebut. Misalnya, fungsi matematika seperti penjumlahan atau perhitungan lainnya dapat menghasilkan nilai yang akan digunakan di tempat lain dalam program.

Jenis Fungsi	Contoh Kode	Keterangan
Fungsi dengan keyword ‘return’	`def tambah(a, b):` `  return a + b`	Fungsi ini **menghasilkan nilai** dari sebuah proses (misalnya, hasil perhitungan). Nilai ini bisa disimpan dalam variabel untuk digunakan lagi.
Fungsi tanpa keyword ‘return’	`def printHello():` `  print("Hello")`	Fungsi ini **melakukan sebuah aksi** (misalnya, menampilkan teks di layar). Tujuannya bukan untuk menghasilkan nilai, melainkan untuk menjalankan perintah.

2. Pemanggilan Function

Setelah membuat fungsi di Python menggunakan sintaks `**def**`, kita dapat memanggilnya dengan menuliskan nama fungsi diikuti oleh tanda kurung yang berisi parameter (jika ada) yang diperlukan. Penting untuk memastikan bahwa argumen yang diberikan sesuai dengan tipe data yang diharapkan oleh fungsi.

```
def printHello():
    print("Hello, world!")

printHello()
✓ 0.0s
Hello, world!
```

3. Function dengan Parameter

Fungsi dapat menerima nilai dari luar, yang disebut **parameter**. Parameter ini digunakan untuk menentukan apa yang akan dilakukan fungsi.

a. Default Argument (Argumen Default)

Default argument adalah nilai bawaan yang diberikan kepada parameter dalam definisi fungsi. Jika nilai untuk parameter tersebut tidak disediakan saat pemanggilan fungsi, maka nilai default akan digunakan.

```python
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet()
greet("Peter Parker")
```

✓ 0.0s

```
Hello, Guest!
Hello, Peter Parker!
```

b. Keyword Arguments (Named Arguments)

Keyword arguments memungkinkan untuk memanggil fungsi dengan menyebutkan nama parameter beserta nilainya, tanpa harus mengikuti urutan yang didefinisikan dalam fungsi. Artinya, kita bisa mengisi parameter fungsi tidak mengikuti urutan yang ditentukan pada suatu fungsi.

```python
def describe_pet(name, animal_type):
    print(f"{name} is a {animal_type}.")

describe_pet(animal_type="dog", name="Buddy")
```

✓ 0.0s

```
Buddy is a dog.
```

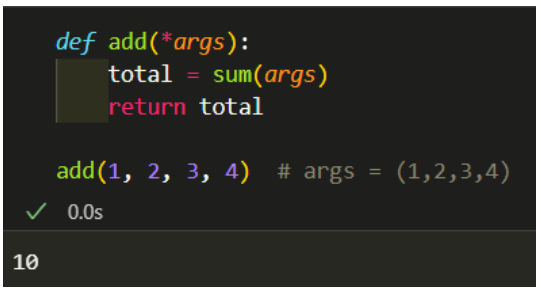
c. Positional Arguments (Argumen Posisi)

Positional arguments adalah argument yang diberikan berdasarkan urutan parameter yang telah didefinisikan dalam fungsi. Misalnya pada fungsi dibawah ini. Urutan parameternya adalah “name” dan “age”. Maka dari itu, kita harus mengisi argument saat pemanggilan fungsi dengan urutan yang sama.

``python def deskripsi_karakter(nama, umur): print(f"Nama: {nama}, Umur: {umur}") deskripsi_karakter("Bob", 20) ``` ✓ 0.0s ``` Nama: Bob, Umur: 20 ```	```python def deskripsi_karakter(nama, umur): print(f"Nama: {nama}, Umur: {umur}") deskripsi_karakter(20, "Bob") ``` ✓ 0.0s ``` Nama: 20, Umur: Bob ```
Pemanggilan Benar	**Pemanggilan Salah**

d. Arbitrary Arguments (Variable-Length Arguments *args and **kwargs)

Arbitrary Arguments (*args dan **kwargs) memungkinkan kita untuk membuat fungsi yang bisa menerima banyak nilai tanpa batasan jumlahnya.

Jenis	Contoh Kode	Keterangan
*args		***args*** mengumpulkan nilai-nilai tersebut sebagai grup dalam bentuk **tupel**,
**kwargs		****kwargs*** akan mengumpulkan nilai-nilai tersebut dalam bentuk **dictionary** yang memungkinkan kita memberi nama kepada setiap nilainya saat memanggil fungsi.
Kombinasi antara *args dengan **kwargs		

4. Function Rekursif

Fungsi rekursif adalah fungsi yang memiliki kemampuan untuk memanggil dirinya sendiri dalam *body* atau tubuhnya. Penggunaan rekursif dapat menghasilkan perulangan terbatas yang berhenti ketika kondisi tertentu terpenuhi, atau dapat juga

mengakibatkan perulangan tanpa batas yang dapat menyebabkan *RecursionError* jika tidak dikelola dengan benar. Dalam rekursif terbatas, fungsi akan memanggil dirinya dengan parameter yang berubah pada setiap pemanggilan hingga mencapai kondisi akhir yang diinginkan.

<table>
<tr>
<td>

```
def show_intervals(start, end):
    print(start, end="")
    start = start + 1
    if start <= end:
        show_intervals(start, end)

show_intervals(1, 10)
```

✓ 0.0s

```
12345678910
```

</td>
<td>

```
def show_multiple(value):
    print(value)
    value = value + value
    show_multiple(value)

show_multiple(2)
# Silahkan dicoba pada laptop
masing-masing
```

</td>
</tr>
<tr>
<td>Contoh rekursif terbatas</td>
<td>Contoh rekursif tanpa batas</td>
</tr>
</table>

5. Type Hinting pada Function

Type hinting adalah cara memberikan informasi tentang tipe data yang diharapkan dari parameter dan nilai kembali fungsi. Meskipun Python adalah bahasa yang dinamis, penggunaan *type hinting* dapat membantu dalam dokumentasi dan pemahaman kode.

```
def divide(a: float, b: float) -> float:
    return a / b

result = divide(10.0, 2.0)
result
```

✓ 0.0s

```
5.0
```

Dalam contoh di atas, a: float dan b: float adalah *type hint* untuk parameter, sedangkan -> float adalah *type hint* untuk nilai `**return**` dari fungsi.

6. Docstring dalam Function

Docstring adalah string yang ditempatkan di awal badan (*body*) sebuah fungsi dalam bahasa pemrograman Python. Kegunaannya adalah untuk menyediakan panduan, penjelasan, atau dokumentasi tentang cara penggunaan dan tujuan fungsi

tersebut. `**help()**` adalah alat yang berguna untuk mengakses dan melihat isi dari *docstring* yang telah didefinisikan dalam sebuah fungsi. Ini memungkinkan pengguna atau pengembang untuk mendapatkan pemahaman yang lebih baik tentang fungsi tersebut tanpa harus merujuk ke kode sumber asli.

```
def docString():
    """
    Ini adalah contoh dari Docstring
    """
    pass

help(docString)
```

✓ 0.0s

```
Help on function docString in module __main__:

docString()
    Ini adalah contoh dari Docstring
```

B. Error Handling

1. Penggunaan Try-Except

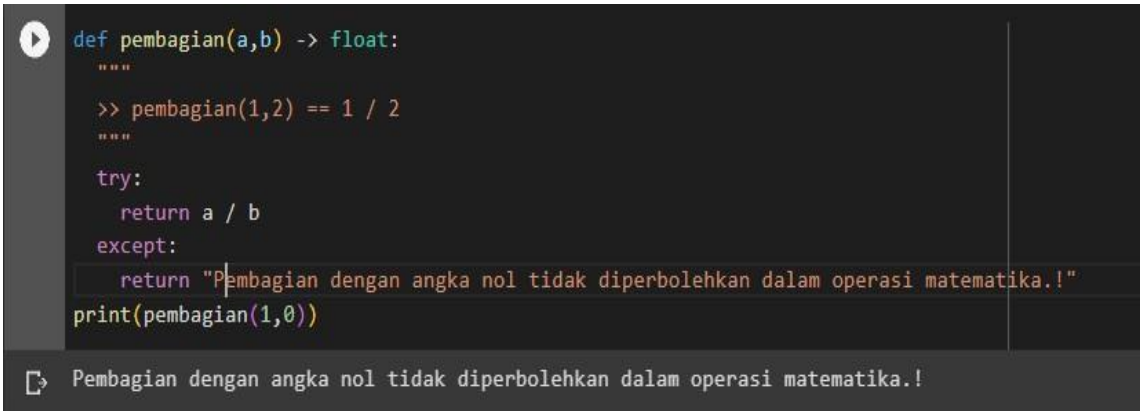
Pernyataan `**try**` digunakan untuk menjalankan blok kode yang mungkin menghasilkan masalah atau kesalahan. Jika terjadi masalah di dalam blok `**try**`, maka pernyataan `**except**` akan merespons dan menjalankan tindakan koreksi yang sesuai. Dengan begitu, program dapat mengatasi masalah dan melanjutkan eksekusi tanpa menghentikan program secara tiba-tiba.

```
def pembagian(a,b) -> float:
    return a / b
print(pembagian(1,0))
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-f84a4ea9a1fd> in <cell line: 3>()
      1 def pembagian(a,b) -> float:
      2     return a / b
----> 3 print(pembagian(1,0))

<ipython-input-3-f84a4ea9a1fd> in pembagian(a, b)
      1 def pembagian(a,b) -> float:
----> 2     return a / b
      3 print(pembagian(1,0))

ZeroDivisionError: division by zero

SEARCH STACK OVERFLOW
```

Pada dasarnya, pernyataan `try` berfungsi sebagai perisai yang melindungi program dari berhenti akibat *error*. Misalnya, jika program mencoba membagi angka oleh nol,

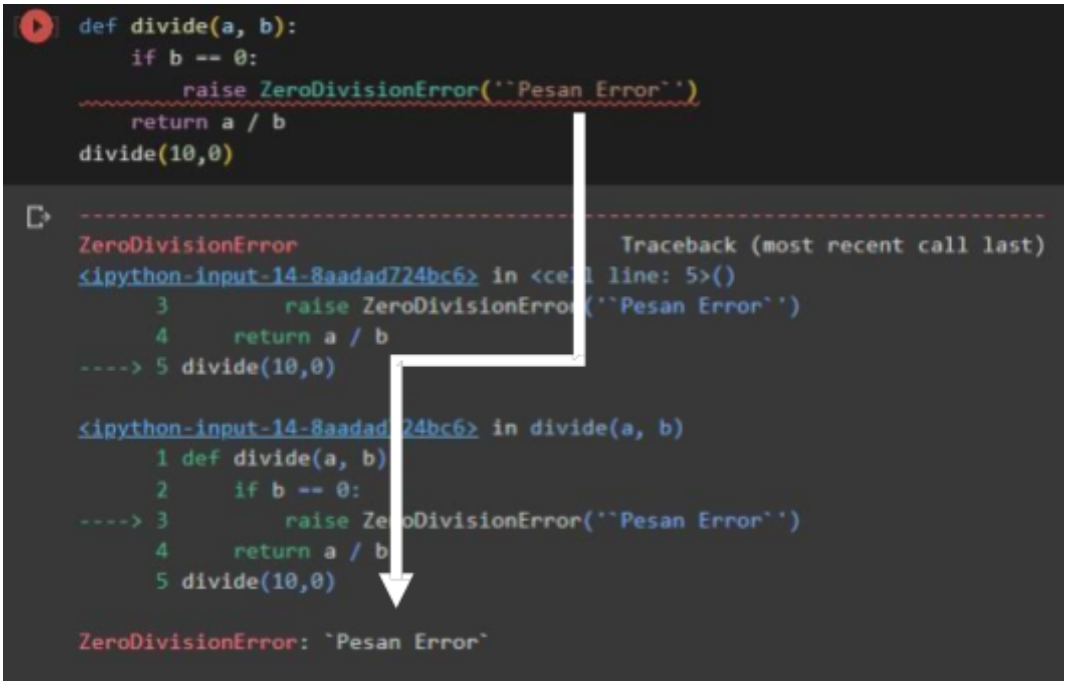
pengecualian *ZeroDivisionError* akan muncul. Namun, dengan menggunakan pernyataan `try` dan `except`, kita dapat menangani kasus ini dengan memberikan pesan yang lebih informatif kepada pengguna dan mencegah program dari berhenti secara tiba- tiba. Dengan kata lain, penanganan pengecualian memungkinkan kita untuk merencanakan "cadangan" dalam kode kita, sehingga program dapat berjalan dengan lebih lancar meskipun ada masalah yang muncul.



Penggunaan pernyataan `except` secara khusus memungkinkan kita untuk merespons pengecualian tertentu dengan cara yang sesuai. Misalnya, jika kita ingin menangani kasus pembagian dengan angka nol, kita dapat menggunakan pernyataan `except ZeroDivisionError` untuk menangkap hanya pengecualian jenis tersebut.

2. Penggunaan Raise untuk Memicu Exception

Keyword `**raise**` digunakan untuk memicu exception secara manual dalam Python. Misalnya, Anda dapat menggunakan `**raise**` untuk menunjukkan situasi khusus yang memerlukan penanganan exception tertentu. Dengan cara ini, Anda dapat memberikan informasi yang lebih spesifik dan relevan tentang kesalahan atau masalah yang mungkin terjadi dalam program Anda, memungkinkan Anda untuk melakukan penanganan yang sesuai dan memecahkan masalah dengan lebih efektif.



3. Penggunaan Else dan Finally

Blok `**else**` dalam pernyataan `**try**` digunakan untuk menjalankan kode jika tidak ada kesalahan (exception) yang terjadi. Sementara itu, blok `finally` akan selalu dijalankan setelah blok `**try**`, `**except**`, dan blok `**else**` selesai, tanpa memperhatikan apakah ada kesalahan atau tidak. Hal ini berguna untuk melakukan tindakan akhir, seperti membersihkan sumber daya, setelah proses dalam blok `**try**` dan blok `**else**‘ selesai, baik itu berhasil atau tidak. Dengan kata lain, blok `**finally**` adalah tempat yang tepat untuk menempatkan kode yang harus dijalankan akhir *regardless of what happens*.

```
try:
    angka1 = 10
    angka2 = 1
    hasil = angka1 / angka2
except ZeroDivisionError:
    print("Error: Pembagian dengan angka nol tidak diizinkan.")
else:
    print("Hasil pembagian:", hasil) # Blok ini akan dijalanlan
                                     # karena except tidak terpenuhi
finally:
    print("Proses selesai.")
```

```
Hasil pembagian: 10.0
Proses selesai.
```

C. RegEx

RegEx atau Regular Expression adalah susunan karakter yang mendefinisikan suatu pattern yang ingin dicari. Umumnya penggunaan regex dapat kita temukan saat melakukan pengecekan inputan apakah termasuk email yang valid atau tidak. Di dalam bahasa pemrograman Python, kita dapat menggunakan module re untuk bekerja menggunakan regex. Contoh penggunaan regex dengan menggunakan module re dapat dilihat pada contoh berikut:

```
import re

pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)

if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")
```

Fungsi `re.match()` untuk mengetes apakah variabel `test_string` dimulai dengan huruf a dan diakhiri dengan huruf s serta terdiri dari 5 huruf. Fungsi tersebut akan mengembalikan sebuah object match jika patternnya sesuai. Jika tidak maka akan mengembalikan nilai `None`.

1. MetaCharacters

MetaCharacters merupakan karakter yang diinterpretasikan secara khusus oleh RegEx engine. Berikut adalah list dari metacharacters:

a. [] – Square Brackets

Simbol **[]** menentukan sebuah set karakter yang ingin dicocokkan.

Ekspresi	String	Cocok ?
[abc]	a	1 cocok
	ac	2 cocok
	Hello	Tidak ada cocok
	abc de ca	5 cocok

Di sini, **[abc]** akan mencocokkan jika sebuah string berisi salah satu dari a, b, atau c.

Kita juga dapat menentukan range dari character dengan menggunakan **-**.

Sebagai contoh:

- [a-e] sama dengan [abcde]
- [1-4] sama dengan [1234]
- [0,3] sama dengan [0123]

Dan kita juga dapat menggunakan symbol **^.** Contoh:

- [^abc] yang berarti semua karakter selain **a** atau **b** atau **c**
- [^0-9] yang berarti semua karakter non-digit

b. . – Period

Simbol **.** akan mencocokkan semua karakter kecuali baris baru.

Ekspresi	String	Cocok ?
..	a	Tidak ada cocok
	ac	1 cocok
	acd	1 cocok
	abde	2 cocok

Di sini, **..** akan mencocokkan jika sebuah string berisi minimal 2 karakter.

c. ^ – Caret

Simbol ^ digunakan untuk mengecek jika sebuah string dimulai oleh karakter tertentu.

Ekspresi	String	Cocok ?
^a	a	1 cocok
	abc	1 cocok
	bac	Tidak ada cocok
^ab	abc	1 cocok
	acb	Tidak ada cocok (dimulai dengan a tapi tidak diikuti oleh b)

d. $ – Dollar

Simbol **$** digunakan untuk mengecek jika sebuah string diakhiri oleh karakter tertentu.

Ekspresi	String	Cocok ?

a$	a	1 cocok
	formula	1 cocok
	cab	Tidak ada cocok

e. * – Star

Simbol * akan mencocokkan nol atau lebih kemunculan pola yang tersisa.

Ekspresi	String	Cocok ?
ma*n	mn	1 cocok
	man	1 cocok
	maaan	1 cocok
	main	Tidak cocok (a tidak diikuti oleh n)
	woman	1 cocok

f. + – Plus

Simbol + akan mencocokkan satu atau lebih kemunculan pola yang tersisa.

Ekspresi	String	Cocok ?
ma+n	mn	Tidak cocok (tidak ada karakter **a**)
	man	1 cocok
	maaan	1 cocok
	main	Tidak cocok (**a** tidak diikuti oleh **n**)
	woman	1 cocok

g. ? – Question Mark

Simbol **?** akan mencocokkan nol atau satu kemunculan string sebelum pola pada regex.

Ekspresi	String	Cocok ?
	mn	1 cocok

ma?n	man	1 cocok
	maaan	Tidak cocok (lebih dari satu karakter **a**)
	main	Tidak cocok (**a** tidak diikuti oleh **n**)
	woman	1 cocok

h. {m, n} – Braces

RegEx ini akan mencari repetisi setidaknya **m** dan sebanyak **n** dari pola yang tersisa.

Ekspresi	String	Cocok ?
a{2,3}	abc dat	Tidak cocok
	abc daat	1 cocok (pada **d<u>aa</u>t**)
	aabc daaat	2 cocok (pada **<u>aa</u>bc** dan **d<u>aaa</u>t**)
	aabc daaaat	2 cocok (pada **<u>aa</u>bc** dan **d<u>aaa</u>at**)
[0-9]{2,4}	ab123csde	1 cocok
	12 and 345673	3 cocok (**<u>12</u>**, **<u>3456</u>**, dan **<u>73</u>**)
	1 and 2	Tidak cocok

Pada regex **[0-9]{2,4}** akan mencocokkan sekurangnya 2 digit numerik tapi tidak lebih dari 4 digit numerik.

i. | – Alternation

Simbol | sama seperti operator or.

Ekspresi	String	Cocok ?
a|b	cde	Tidak cocok
	ade	1 cocok (pada **<u>a</u>de**)

	acdbea	3 cocok (pada **acdbea**)

j. () – Group

Simbol **()** digunakan untuk mengelompokkan pola regex. Contoh, **(a|b|c)xz** akan mencocokkan semua string yang memiliki karakter a atau b atau c lalu diikuti oleh xz.

Ekspresi	String	Cocok ?
(a|b|c)xz	ab xy	Tidak cocok
	abxz	1 cocok (pada **abxz**)
	axz cabxz	2 cocok (pada **axz**bc **cabxz**)

k. \ – Backslash

Simbol \ digunakan untuk memastikan bahwa sebuah karakter tidak diperlakukan secara khusus. Sebagai contoh, regex **\$a** akan mencocokkan jika sebuah string memiliki karakter **$** diikuti oleh **a**. Di sini, **$** tidak diinterpretasikan oleh engine RegEx.

2. Special Sequences

Special sequences digunakan untuk mencari sebuah string berdasarkan lokasi di mana string tersebut berada. Special sequences membuat pola yang umum digunakan lebih mudah untuk ditulis.

Special Sequence	Deskripsi	Contoh	
\A	Mengembalikan kecocokan jika karakter yang ditentukan berada di awal string	\Athe	the sun
\b	Mengembalikan kecocokan dimana karakter yang ditentukan	\bfoo	A football
		foo\b	The foo

	berada di awal atau di akhir kata ("r" pada awalnya memastikan bahwa string diperlakukan sebagai "string mentah")		
\B	Mengembalikan kecocokan dimana karakter yang ditentukan ada, tetapi BUKAN di awal (atau di akhir) kata ("r" pada awalnya memastikan bahwa string diperlakukan sebagai "string mentah")	\Bfoo	Afootball
		foo\b	The afootest
\d	Mengembalikan kecocokan dimana string berisi angka (angka dari 0-9)	12abc3	3 cocok (pada **12abc3**)
\D	Mengembalikan kecocokan dimana string TIDAK mengandung angka	1ab34”50	3 cocok (pada **1ab34”50**)
\s	Mengembalikan kecocokan dimana string berisi karakter spasi	Python RegEx	1 cocok
\S	Mengembalikan kecocokan dimana string TIDAK mengandung karakter spasi	a b	2 cocok (pada **a b**)
\w	Mengembalikan kecocokan di mana string berisi karakter kata apa pun (karakter dari a hingga Z, angka dari 0-9, dan garis bawah _ karakter)	12&”: ;c	3 cocok (pada **12&: ;c**)

Tip: Kalian bisa membuat dan menguji regex yang sudah dibuat, kalian dapat menggunakan tools online seperti https://regex101.com/

3. Python RegEx

Python memiliki sebuah modul bernama **re** yang digunakan untuk menjalankan perintah regex. Untuk menggunakannya kita perlu mengimport modul **re**.

```
import re
```

Berikut adalah beberapa function dalam modul **re** untuk bekerja dengan regex,

a. **`re.findall()`**

Fungsi **`re.findall()`** mengembalikan sebuah list string yang berisi semua kecocokan.

```
import re

string = 'Hello 1234. Welcome 4523 Kaito 1412'
pattern = '\d+'

result = re.findall(pattern, string)
print(result)
```

✓ 0.0s

```
['1234', '4523', '1412']
```

Jika tidak terdapat kecocokan, maka akan dikembalikan list kosong.

b. **`re.split()`**

Fungsi **`re.split()`** membelah sebuah string di mana ada kecocokan dan mengembalikannya ke dalam sebuah list string.

```
import re

string = 'Hello 1234. Welcome 4523 Kaito 1412'
pattern = '\d+'

result = re.split(pattern, string)
print(result)
```

✓ 0.0s

```
['Hello ', '. Welcome ', ' Kaito ', '']
```

Jika pola tidak ditemukan, maka akan dikembalikan sebuah list yang berisi string asli.

c. **`re.sub()`**

Fungsi **`re.sub()`** akan mengembalikan sebuah string di mana bagian yang cocok akan diganti dengan konten dari variable **`replace`**.

```
import re

string = 'Hello 1234 Welcome 4523 \n Kaito 1412'
pattern = '\s+'
replace = ''

result = re.sub(pattern, replace, string)
print(result)
```

✓ 0.0s

```
Hello1234Welcome4523Kaito1412
```

Jika pola tidak ditemukan, maka akan dikembalikan string asli.

d. `re.subn()`

Fungsi **`re.subn()`** sama dengan **`re.sub()`** kecuali ini mengembalikan sebuah tuple dari 2 item yang berisi string baru dan jumlah pergantian yang dilakukan.

```
import re

string = 'Hello 1234 Welcome 4523 \n Kaito 1412'
pattern = '\s+'
replace = ''

result = re.subn(pattern, replace, string)
print(result)
```

✓ 0.0s

```
('Hello1234Welcome4523Kaito1412', 5)
```

e. `re.search()`

Fungsi **`re.search()`** akan mencari lokasi pertama di mana pola regex menghasilkan kecocokan dengan string. Jika pencarian berhasil, maka **`re.search()`** akan mengembalikan objek; jika tidak, akan mengembalikan **`None`**.

```python
import re

string = "Hello World"
match = re.search('\AHello', string)

if match:
    print("pattern found inside the string")
else:
    print("pattern not found")
```

✓ 0.0s

```
pattern found inside the string
```