

TASK 3 MODUL 4

Nama: Sedri Sella Jumeni

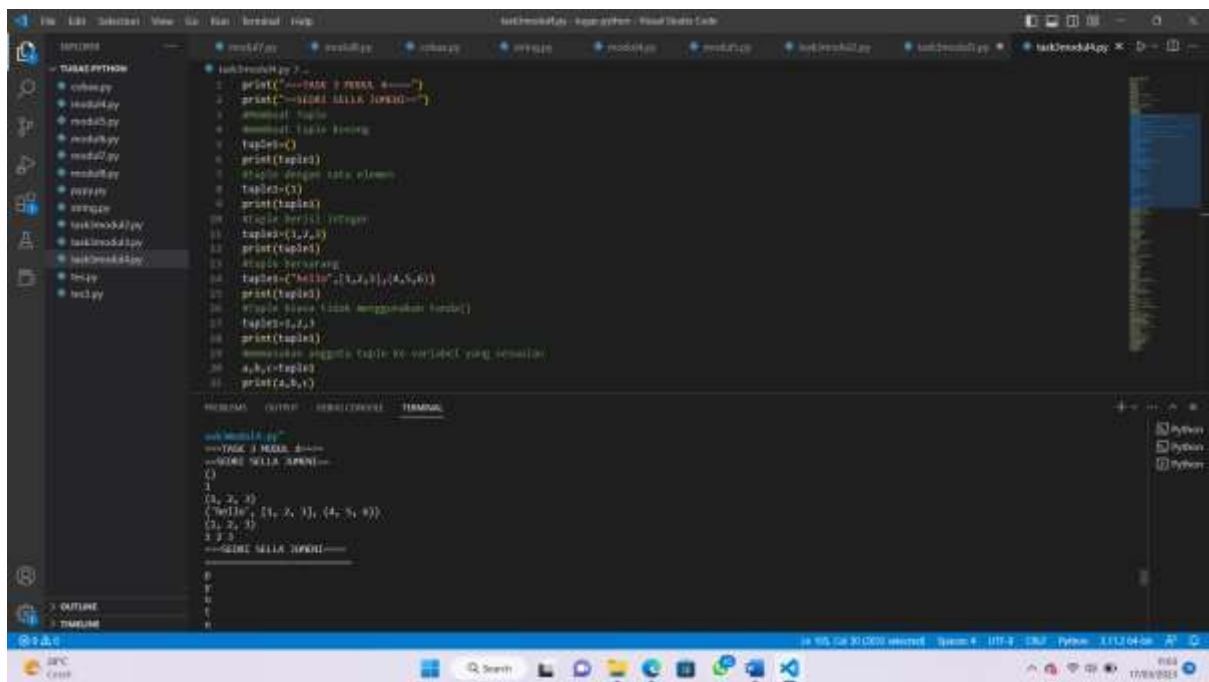
Nim: 211001073

Kelas: Pemrograman Python D

Tuple

Tuple mirip dengan list. Bedanya, tuple bersifat immutable, sehingga anggotanya tidak bisa diubah. **Membuat Tuple** : Tuple dibuat dengan meletakkan semua anggota di dalam tanda kurung (), masing-masing dipisahkan oleh tanda koma.

Contoh:

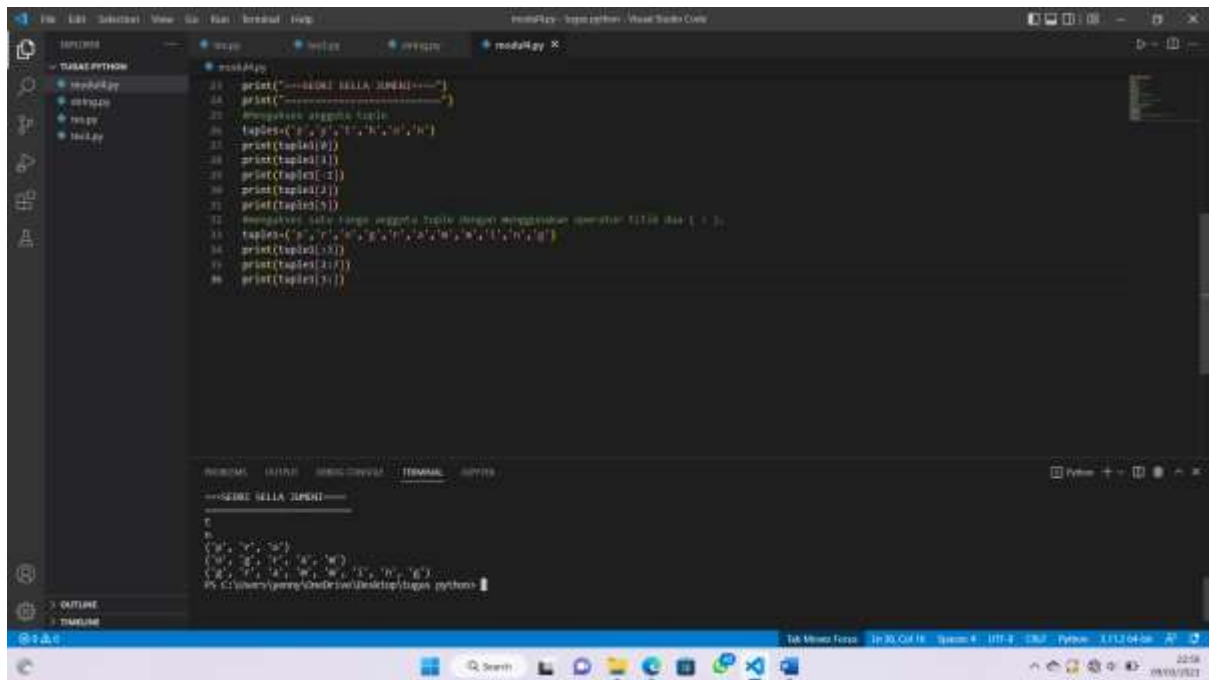


```
1 print("==DATA 3 MEASA 4==")
2 print("==Sedri Sella Jumeni==")
3
4 #membuat tuple kosong
5 tupSel=()
6 print(tupSel)
7 #tuple dengan satu elemen
8 tupSel=(1)
9 print(tupSel)
10 #tuple berisi 3tuplen
11 tupSel=(1,2,3)
12 print(tupSel)
13 #tuple berurutan
14 tupSel=("hello",[1,2,3],[4,5,6])
15 print(tupSel)
16 #tuple hanya tidak menggunakan kurva()
17 tupSel=1,2,3
18 print(tupSel)
19 #membuat anggota tuple ke variabel yang sesuai
20 a,b,c=tupSel
21 print(a,b,c)
```

Mengakses Anggota Tuple: kita bisa mengakses anggota tuple lewat indeksnya menggunakan format `namatuple[indeks]`. Indeks dimulai dari 0 untuk anggota pertama. Selain itu, indeks negatif juga bisa dipakai mulai dari -1 untuk anggota terakhir tuple.

Contoh:

TASK 3 MODUL 4

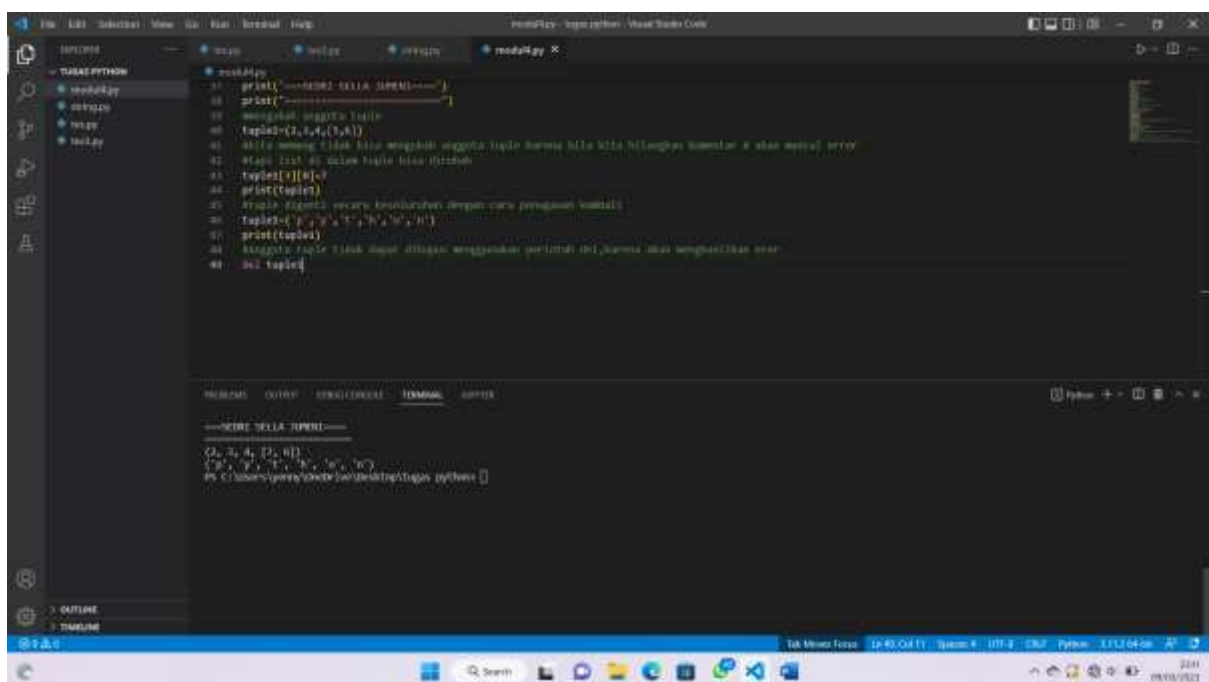


```
21 print("===== SELLA SUPENI =====")
22 print("=====")
23 #membuat anggota tuple
24 tuplex=("a","b","c","d","e","f")
25 print(tuplex[0])
26 print(tuplex[1])
27 print(tuplex[-1])
28 print(tuplex[2])
29 print(tuplex[5])
30 #membuat tuple yang anggota tuple dengan menggunakan operator slicing dan [:]
31 tuplex=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p")
32 print(tuplex[3])
33 print(tuplex[2:5])
34 print(tuplex[5:])
```

```
===== SELLA SUPENI =====
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
PS C:\Users\perry\OneDrive\Desktop>python3 modul4.py
```

Mengubah Anggota Tuple: Setelah tuple dibuat, maka anggota tuple tidak bisa lagi diubah atau dihapus. Akan tetapi, bila anggota tuple-nya adalah tuple bersarang dengan anggota seperti list, maka item pada list tersebut dapat diubah.

Contoh:



```
31 print("===== SELLA SUPENI =====")
32 print("=====")
33 #membuat anggota tuple
34 tuplex=(1,2,3,4,5,6)
35 #jika memang tidak bisa mengubah anggota tuple karena jika kita nilainya komentar & akan muncul error
36 #tapi list di dalam tuple bisa diubah
37 tuplex[3][0]=7
38 print(tuplex)
39 #jika anggota secara keseluruhan dengan cara pengisian kembali
40 tuplex=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p")
41 print(tuplex)
42 #Anggota tuple tidak dapat dihapus menggunakan perintah del, karena akan menghasilkan error
43 del tuplex
```

```
===== SELLA SUPENI =====
(1, 2, 3, 7, 5, 6)
('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p')
PS C:\Users\perry\OneDrive\Desktop>python3 modul4.py
```

TASK 3 MODUL 4

Menguji Keanggotaan Tuple : kita bisa menguji apakah sebuah objek adalah anggota dari tuple atau tidak, yaitu dengan menggunakan operator `in` atau `not in` untuk kebalikannya.

Contoh:

The screenshot shows the Visual Studio Code editor with a file named 'testMap.py' open. The code in the file is as follows:

```

1 # testMap
2
3 print("=====SHELL JUNK=====")
4 print("=====")
5
6 #map() function
7
8 tuplet=('a',1,1,1,'a','b','c','d')
9
10 #unpacking
11
12 print('a in tuple')
13 print('1 in tuple')
14 print('a' in tuple)
15 print('b' in tuple)
16

```

The terminal output shows the execution of the script:

```

=====SHELL JUNK=====
=====
True
False
True
False
C:\Users\jyang\OneDrive\Desktop\Maple-python

```

Iterasi pada Tuple : Kita bisa menggunakan for untuk melakukan iterasi pada tiap anggota dalam tuple.

Contoh:

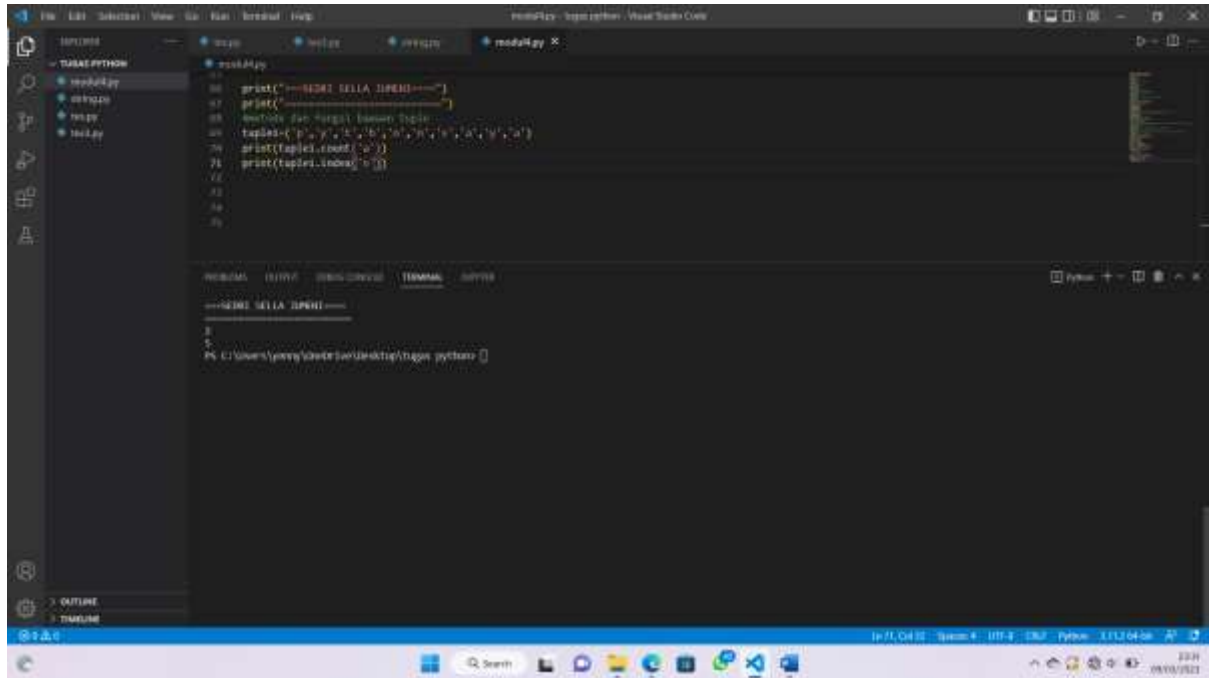
[illegible]

TASK 3 MODUL 4

Metode dan Fungsi Bawaan Tuple : Tuple hanya memiliki dua buah metode yaitu count() dan index().

- Metode count(x) berfungsi mengembalikan jumlah item yang sesuai dengan x pada tuple.
- Metode index(x) berfungsi mengembalikan indeks dari item pertama yang sama dengan x.

Contoh:



```
16 print("===== SELLA SIPENI =====")
17 print("=====")
18 #metode dan fungsi bawaan tuple
19 tuple1=('p','y','t','h','o','n','s','a','n','a')
20 print(tuple1.count('a'))
21 print(tuple1.index('n'))
```

```
===== SELLA SIPENI =====
=====
5
11
```

Set

Set adalah salah satu tipe data di Python yang tidak berurut (unordered). Set memiliki anggota yang unik (tidak ada duplikasi). Set bisa digunakan untuk melakukan operasi himpunan matematika seperti gabungan, irisan, selisih, dan lain - lain.

Membuat Set : Set dibuat dengan meletakkan anggota - anggotanya di dalam tanda kurung kurawal {}, dipisahkan menggunakan tanda koma. Kita juga bisa membuat set dari list dengan memasukkan list ke dalam fungsi set().

TASK 3 MODUL 4

Contoh:

The screenshot displays a Windows 10 desktop environment. The primary focus is the Visual Studio Code (VS Code) application, which is open with a file named 'modul.py'. The editor's interface includes a sidebar on the left with icons for Explorer, Search, Run and Debug, and Extensions. The main editor area shows the following Python code:

```

1  # modul.py
2
3  print("=====")
4  print("=====")
5
6  class Set:
7
8      def __init__(self):
9          self._data = {}
10
11      def add(self, element):
12          self._data[element] = True
13
14      def __str__(self):
15          return str(self._data)
16
17      def __len__(self):
18          return len(self._data)
19
20      def __iter__(self):
21          return iter(self._data)
22
23      def __contains__(self, element):
24          return element in self._data
25
26      def __getitem__(self, element):
27          return self._data[element]
28
29      def __setitem__(self, element, value):
30          self._data[element] = value
31
32      def __delitem__(self, element):
33          del self._data[element]
34
35      def __repr__(self):
36          return repr(self._data)
37
38      def __eq__(self, other):
39          return self._data == other._data
40
41      def __ne__(self, other):
42          return self._data != other._data
43
44      def __lt__(self, other):
45          return self._data < other._data
46
47      def __gt__(self, other):
48          return self._data > other._data
49
50      def __le__(self, other):
51          return self._data <= other._data
52
53      def __ge__(self, other):
54          return self._data >= other._data
55
56      def __and__(self, other):
57          return Set(self._data & other._data)
58
59      def __or__(self, other):
60          return Set(self._data | other._data)
61
62      def __xor__(self, other):
63          return Set(self._data ^ other._data)
64
65      def __sub__(self, other):
66          return Set(self._data - other._data)
67
68      def __add__(self, other):
69          return Set(self._data + other._data)
70
71      def __mul__(self, other):
72          return Set(self._data * other._data)
73
74      def __div__(self, other):
75          return Set(self._data / other._data)
76
77      def __mod__(self, other):
78          return Set(self._data % other._data)
79
80      def __pow__(self, other):
81          return Set(self._data ** other._data)
82
83      def __radd__(self, other):
84          return Set(other._data + self._data)
85
86      def __rsub__(self, other):
87          return Set(other._data - self._data)
88
89      def __rmul__(self, other):
90          return Set(other._data * self._data)
91
92      def __rdiv__(self, other):
93          return Set(other._data / self._data)
94
95      def __rmod__(self, other):
96          return Set(other._data % self._data)
97
98      def __rpow__(self, other):
99          return Set(other._data ** self._data)
100
101      def __iadd__(self, other):
102          self._data |= other._data
103          return self
104
105      def __isub__(self, other):
106          self._data -= other._data
107          return self
108
109      def __imul__(self, other):
110          self._data *= other._data
111          return self
112
113      def __idiv__(self, other):
114          self._data /= other._data
115          return self
116
117      def __imod__(self, other):
118          self._data %= other._data
119          return self
120
121      def __ipow__(self, other):
122          self._data **= other._data
123          return self
124
125      def __iand__(self, other):
126          self._data &= other._data
127          return self
128
129      def __ior__(self, other):
130          self._data |= other._data
131          return self
132
133      def __ixor__(self, other):
134          self._data ^= other._data
135          return self
136
137      def __isub__(self, other):
138          self._data -= other._data
139          return self
140
141      def __iadd__(self, other):
142          self._data += other._data
143          return self
144
145      def __imul__(self, other):
146          self._data *= other._data
147          return self
148
149      def __idiv__(self, other):
150          self._data /= other._data
151          return self
152
153      def __imod__(self, other):
154          self._data %= other._data
155          return self
156
157      def __ipow__(self, other):
158          self._data **= other._data
159          return self
160
161      def __iand__(self, other):
162          self._data &= other._data
163          return self
164
165      def __ior__(self, other):
166          self._data |= other._data
167          return self
168
169      def __ixor__(self, other):
170          self._data ^= other._data
171          return self
172
173      def __isub__(self, other):
174          self._data -= other._data
175          return self
176
177      def __iadd__(self, other):
178          self._data += other._data
179          return self
180
181      def __imul__(self, other):
182          self._data *= other._data
183          return self
184
185      def __idiv__(self, other):
186          self._data /= other._data
187          return self
188
189      def __imod__(self, other):
190          self._data %= other._data
191          return self
192
193      def __ipow__(self, other):
194          self._data **= other._data
195          return self
196
197      def __iand__(self, other):
198          self._data &= other._data
199          return self
200
201      def __ior__(self, other):
202          self._data |= other._data
203          return self
204
205      def __ixor__(self, other):
206          self._data ^= other._data
207          return self
208
209      def __isub__(self, other):
210          self._data -= other._data
211          return self
212
213      def __iadd__(self, other):
214          self._data += other._data
215          return self
216
217      def __imul__(self, other):
218          self._data *= other._data
219          return self
220
221      def __idiv__(self, other):
222          self._data /= other._data
223          return self
224
225      def __imod__(self, other):
226          self._data %= other._data
227          return self
228
229      def __ipow__(self, other):
230          self._data **= other._data
231          return self
232
233      def __iand__(self, other):
234          self._data &= other._data
235          return self
236
237      def __ior__(self, other):
238          self._data |= other._data
239          return self
240
241      def __ixor__(self, other):
242          self._data ^= other._data
243          return self
244
245      def __isub__(self, other):
246          self._data -= other._data
247          return self
248
249      def __iadd__(self, other):
250          self._data += other._data
251          return self
252
253      def __imul__(self, other):
254          self._data *= other._data
255          return self
256
257      def __idiv__(self, other):
258          self._data /= other._data
259          return self
260
261      def __imod__(self, other):
262          self._data %= other._data
263          return self
264
265      def __ipow__(self, other):
266          self._data **= other._data
267          return self
268
269      def __iand__(self, other):
270          self._data &= other._data
271          return self
272
273      def __ior__(self, other):
274          self._data |= other._data
275          return self
276
277      def __ixor__(self, other):
278          self._data ^= other._data
279          return self
280
281      def __isub__(self, other):
282          self._data -= other._data
283          return self
284
285      def __iadd__(self, other):
286          self._data += other._data
287          return self
288
289      def __imul__(self, other):
290          self._data *= other._data
291          return self
292
293      def __idiv__(self, other):
294          self._data /= other._data
295          return self
296
297      def __imod__(self, other):
298          self._data %= other._data
299          return self
300
301      def __ipow__(self, other):
302          self._data **= other._data
303          return self
304
305      def __iand__(self, other):
306          self._data &= other._data
307          return self
308
309      def __ior__(self, other):
310          self._data |= other._data
311          return self
312
313      def __ixor__(self, other):
314          self._data ^= other._data
315          return self
316
317      def __isub__(self, other):
318          self._data -= other._data
319          return self
320
321      def __iadd__(self, other):
322          self._data += other._data
323          return self
324
325      def __imul__(self, other):
326          self._data *= other._data
327          return self
328
329      def __idiv__(self, other):
330          self._data /= other._data
331          return self
332
333      def __imod__(self, other):
334          self._data %= other._data
335          return self
336
337      def __ipow__(self, other):
338          self._data **= other._data
339          return self
340
341      def __iand__(self, other):
342          self._data &= other._data
343          return self
344
345      def __ior__(self, other):
346          self._data |= other._data
347          return self
348
349      def __ixor__(self, other):
350          self._data ^= other._data
351          return self
352
353      def __isub__(self, other):
354          self._data -= other._data
355          return self
356
357      def __iadd__(self, other):
358          self._data += other._data
359          return self
360
361      def __imul__(self, other):
362          self._data *= other._data
363          return self
364
365      def __idiv__(self, other):
366          self._data /= other._data
367          return self
368
369      def __imod__(self, other):
370          self._data %= other._data
371          return self
372
373      def __ipow__(self, other):
374          self._data **= other._data
375          return self
376
377      def __iand__(self, other):
378          self._data &= other._data
379          return self
380
381      def __ior__(self, other):
382          self._data |= other._data
383          return self
384
385      def __ixor__(self, other):
386          self._data ^= other._data
387          return self
388
389      def __isub__(self, other):
390          self._data -= other._data
391          return self
392
393      def __iadd__(self, other):
394          self._data += other._data
395          return self
396
397      def __imul__(self, other):
398          self._data *= other._data
399          return self
400
401      def __idiv__(self, other):
402          self._data /= other._data
403          return self
404
405      def __imod__(self, other):
406          self._data %= other._data
407          return self
408
409      def __ipow__(self, other):
410          self._data **= other._data
411          return self
412
413      def __iand__(self, other):
414          self._data &= other._data
415          return self
416
417      def __ior__(self, other):
418          self._data |= other._data
419          return self
420
421      def __ixor__(self, other):
422          self._data ^= other._data
423          return self
424
425      def __isub__(self, other):
426          self._data -= other._data
```

Mengubah Anggota Set : Set bersifat mutable. Tapi, karena set adalah tipe data tidak berurut (unordered), maka kita tidak bisa menggunakan indeks. Set tidak mendukung indeks ataupun slicing.

Contoh:

The screenshot shows a Visual Studio Code editor with a Python file named `main.py`. The code defines a list `set_saya` and prints its contents at various points during its modification.

```

1  print("=====SET SET LIA. JENEN=====")
2  print("=====")
3  mengahab pengira set
4  membuat set baru
5  set_saya=(1,2,3,4)
6  print(set_saya)
7
8  menambah satu anggota plus data baru
9  set_saya.add(5)
10 print(set_saya)
11 menambahkan beberapa anggota
12 set_saya.update([4,5,6,7,8])
13 print(set_saya)

```

The terminal output shows the execution of the script, displaying the list `{1, 2, 3, 4}`, `{1, 2, 3, 4, 5}`, and `{1, 2, 3, 4, 5, 6, 7, 8}` at the corresponding print statements.

Menghapus Anggota : Set Kita bisa menghapus anggota set dengan menggunakan fungsi `discard()` dan `remove()`.

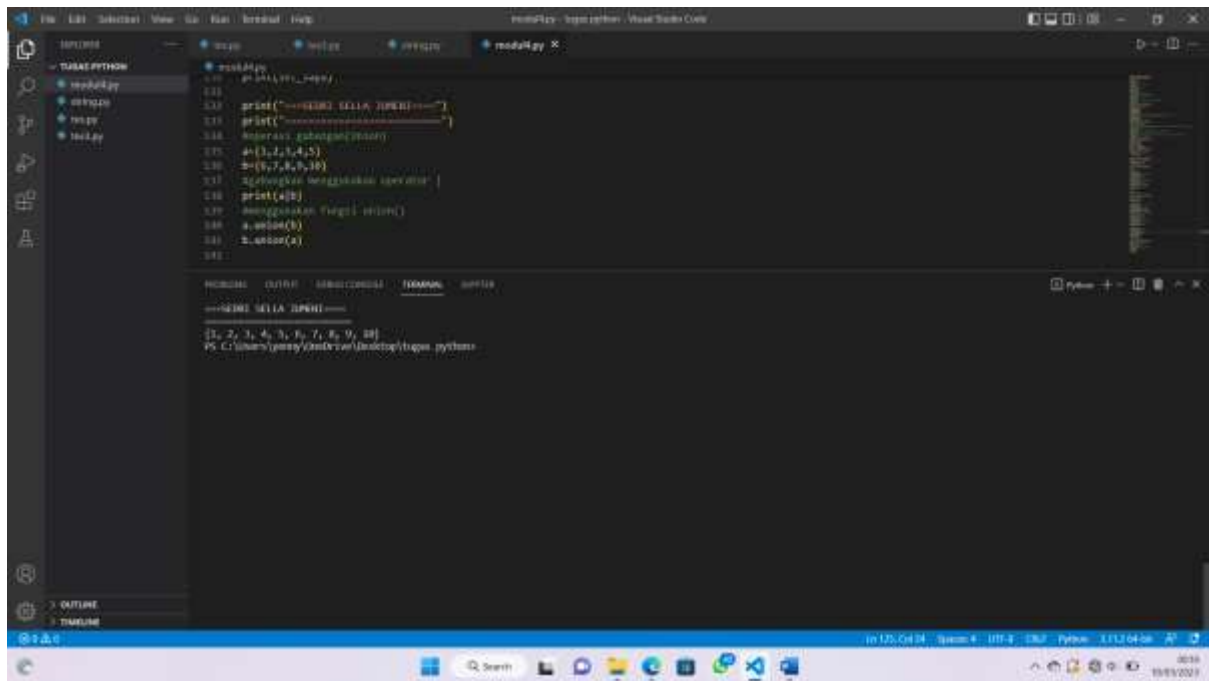
Contoh:



Operasi Gabungan (Union): Gabungan (union) dari A dan B adalah himpunan atau set anggota yang ada di A dan B. Gabungan dapat dibuat dengan menggunakan operator palang (\cup). Selain itu juga bisa dilakukan dengan menggunakan fungsi `union()`.

Contoh:

TASK 3 MODUL 4



```
1 # -*- coding: utf-8 -*-
2 # Author: [Your Name]
3 # Date: [Your Date]
4 # Description: Set operations using Python
5
6 # Define sets A and B
7 A = {1, 2, 3, 4, 5}
8 B = {5, 7, 8, 9, 10}
9
10 # Perform operations
11 # Union of A and B
12 print("Union of A and B")
13 print(A.union(B))
14
15 # Intersection of A and B
16 print("Intersection of A and B")
17 print(A.intersection(B))
18
19 # Difference of A and B
20 print("Difference of A and B")
21 print(A.difference(B))
22 print(B.difference(A))
23
24 # Symmetric Difference of A and B
25 print("Symmetric Difference of A and B")
26 print(A.symmetric_difference(B))
27
28 # Output
29 print("Output:")
30 print(A.union(B))
31 print(A.intersection(B))
32 print(A.difference(B))
33 print(B.difference(A))
34 print(A.symmetric_difference(B))
35
36 # End of script
```

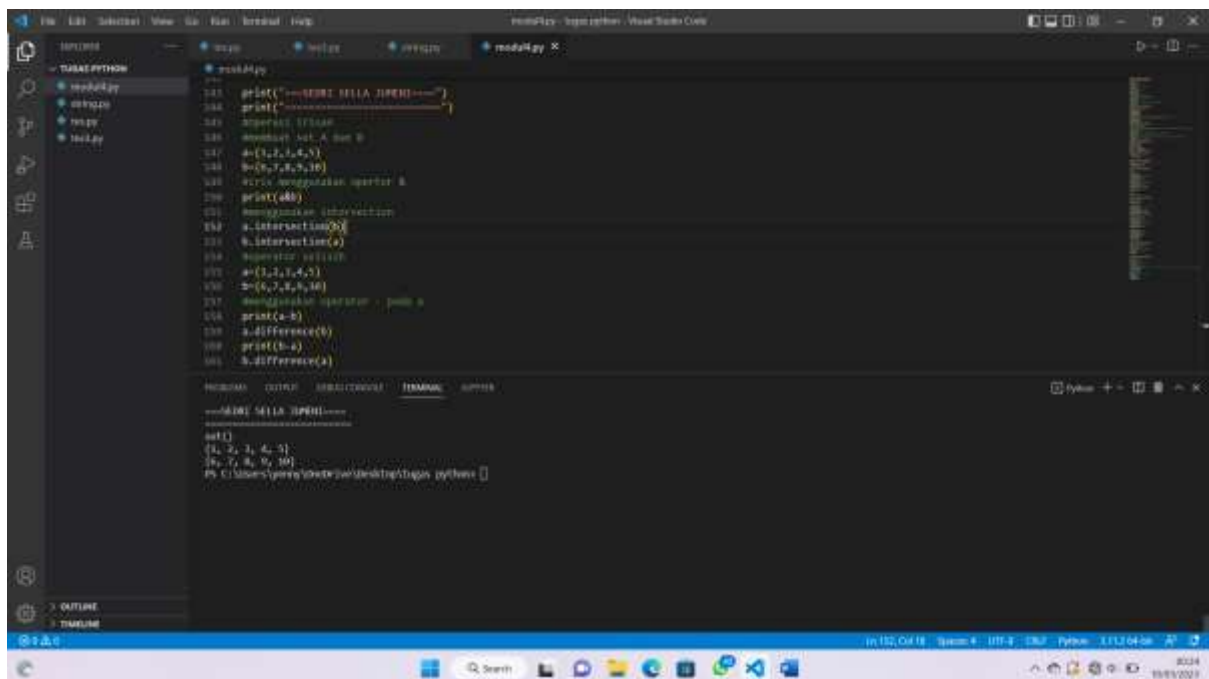
Output:

```
Union of A and B
{1, 2, 3, 4, 5, 7, 8, 9, 10}
Intersection of A and B
{5}
Difference of A and B
{1, 2, 3, 4}
{7, 8, 9, 10}
Symmetric Difference of A and B
{1, 2, 3, 4, 7, 8, 9, 10}
```

Operasi Irisan (Intersection): B. Irisan dilakukan dengan menggunakan operator jangkar (&). Irisan juga bisa dilakukan dengan menggunakan fungsi intersection().

Operasi Selisih (Difference): Selisih dilakukan dengan menggunakan operator kurang (-). Bisa juga dengan menggunakan fungsi difference().

Contoh:



```
1 # -*- coding: utf-8 -*-
2 # Author: [Your Name]
3 # Date: [Your Date]
4 # Description: Set operations using Python
5
6 # Define sets A and B
7 A = {1, 2, 3, 4, 5}
8 B = {5, 7, 8, 9, 10}
9
10 # Perform operations
11 # Union of A and B
12 print("Union of A and B")
13 print(A.union(B))
14
15 # Intersection of A and B
16 print("Intersection of A and B")
17 print(A.intersection(B))
18
19 # Difference of A and B
20 print("Difference of A and B")
21 print(A.difference(B))
22 print(B.difference(A))
23
24 # Symmetric Difference of A and B
25 print("Symmetric Difference of A and B")
26 print(A.symmetric_difference(B))
27
28 # Output
29 print("Output:")
30 print(A.union(B))
31 print(A.intersection(B))
32 print(A.difference(B))
33 print(B.difference(A))
34 print(A.symmetric_difference(B))
35
36 # End of script
```

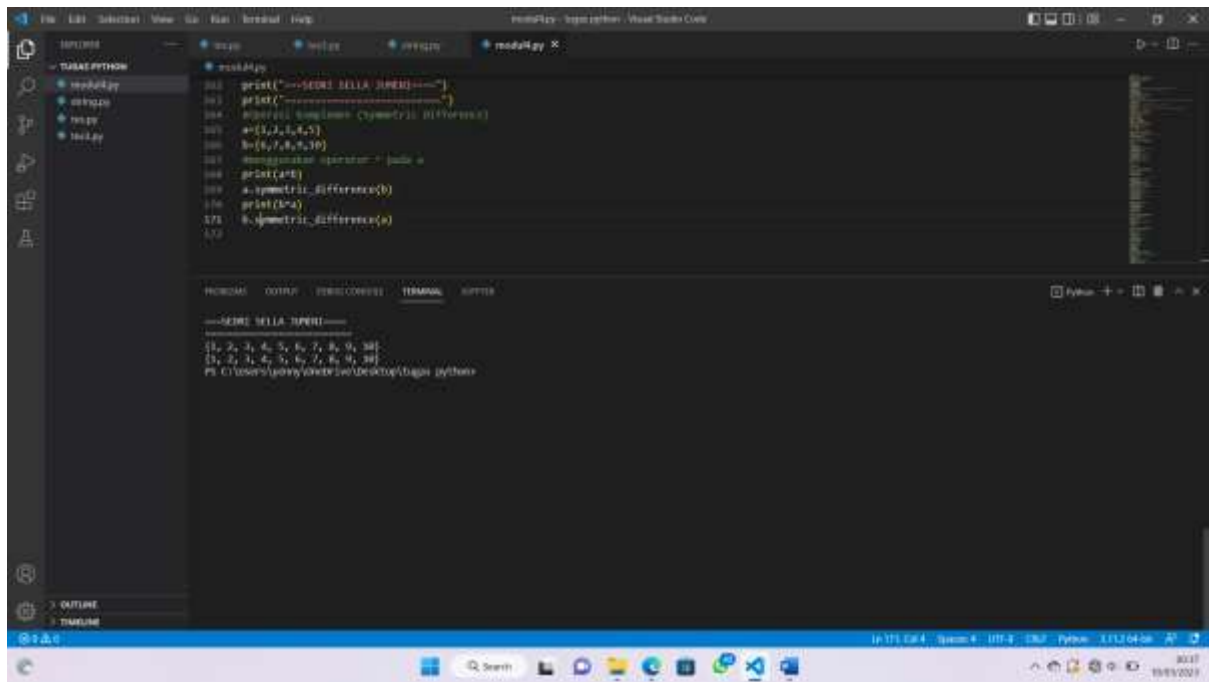
Output:

```
Union of A and B
{1, 2, 3, 4, 5, 7, 8, 9, 10}
Intersection of A and B
{5}
Difference of A and B
{1, 2, 3, 4}
{7, 8, 9, 10}
Symmetric Difference of A and B
{1, 2, 3, 4, 7, 8, 9, 10}
```

Operasi Komplemen (Symmetric Difference): himpunan atau set anggota yang ada di A dan di B, tapi tidak di keduanya. Komplemen dilakukan dengan menggunakan operator ^. Bisa juga dengan menggunakan fungsi symmetric_difference().

Contoh:

TASK 3 MODUL 4



```
142 print("---SEKELI BELLA RUMAH---")
143 print("-----")
144 #Membuat himpunan (symmetric difference)
145 a=[1,2,3,4,5]
146 b=[6,7,8,9,10]
147 #Menggabungkan operator * pada a
148 print(a*b)
149 a.symmetric_difference(b)
150 print(a*b)
151 b.symmetric_difference(a)
152
```

Terminal output:

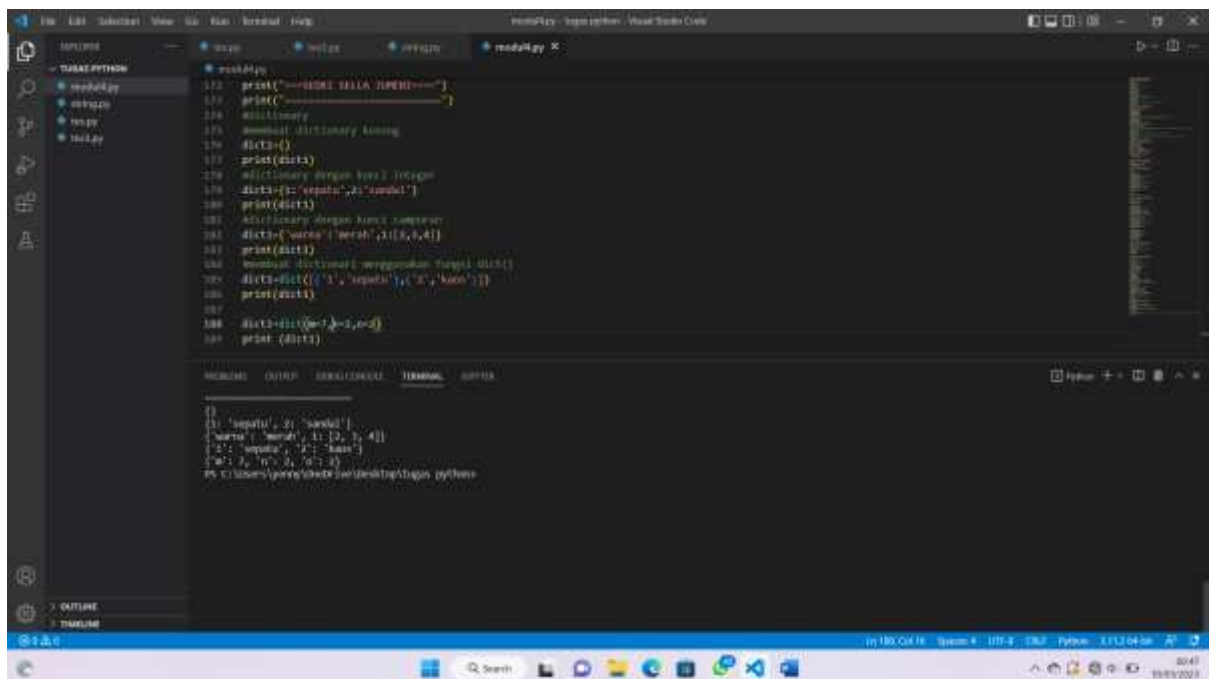
```
---SEKELI BELLA RUMAH---
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\jenny\OneDrive\Desktop>python
```

Dictionary

Dictionary adalah tipe data yang anggotanya terdiri dari pasangan kunci:nilai (key:value). Dictionary bersifat tidak berurut (unordered) sehingga anggotanya tidak memiliki indeks.

Membuat Dictionary: Dictionary dibuat dengan menempatkan anggotanya di dalam tanda kurung kurawal {}, dipisahkan oleh tanda koma.

Contoh:



```
172 print("---SEKELI BELLA RUMAH---")
173 print("-----")
174 #Dictionary
175 #Membuat Dictionary kosong
176 dict1={}
177 print(dict1)
178 #Membuat Dictionary dengan kunci integer
179 dict1={'sepatu':1,'sandal':1}
180 print(dict1)
181 #Membuat Dictionary dengan kunci campuran
182 dict1={'warna':'merah',1:[1,2,3,4]}
183 print(dict1)
184 #Membuat Dictionary menggunakan fungsi dict()
185 dict1=dict(['1','sepatu'],1,'X','kano')
186 print(dict1)
187
188 dict1=dict({'a':1,'b':2,'c':3})
189 print(dict1)
```

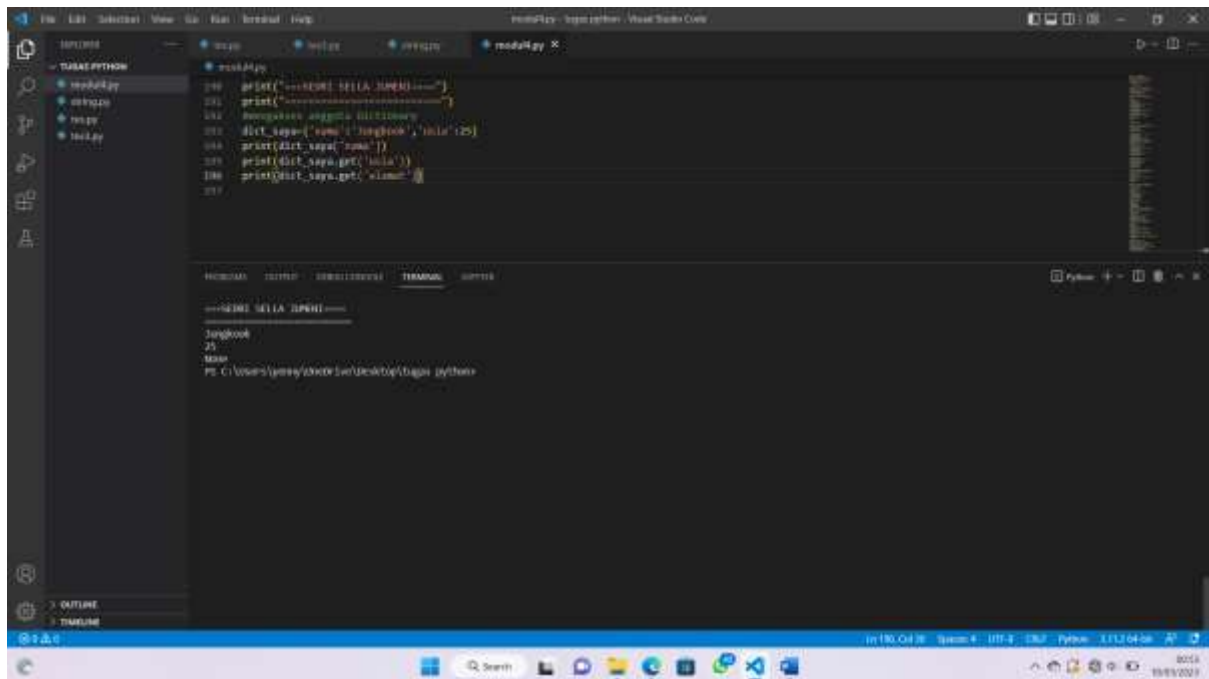
Terminal output:

```
{ }
{'sepatu': 1, 'sandal': 1}
{'warna': 'merah', 1: [1, 2, 3, 4]}
{'1': 'sepatu', 'X': 'kano'}
{'a': 1, 'b': 2, 'c': 3}
PS C:\Users\jenny\OneDrive\Desktop>python
```

Mengakses Anggota Dictionary: Dictionary tidak menggunakan indeks. Anggota dictionary diakses dengan menggunakan kuncinya. Selain itu, bisa juga diakses dengan menggunakan fungsi get().

TASK 3 MODUL 4

Contoh:



```
190 print("===== SELLA JUPHI =====")
191 print("=====")
192
193 #mengases anggota dictionary
194 dict_sapa={"sapa":'Jungbook', 'usia':25}
195 print(dict_sapa['nama'])
196 print(dict_sapa.get('usia'))
197 print(dict_sapa.get('alamat'))
198
```

```
===== SELLA JUPHI =====
=====
Jungbook
25
None
PS C:\Users\jenny\source\repos\Task3\Task3\python>
```

Mengubah Anggota Dictionary Dictionary: bersifat mutable. Kita bisa menambahkan atau mengubah nilai dari anggotanya menggunakan operator penugasan. Bila kunci sudah ada, maka nilainya yang akan diupdate. Bila kunci belum ada, maka akan ditambahkan sebagai kunci baru.

Menghapus Anggota Dictionary: Kita bisa menghapus anggota tertentu pada dictionary dengan menggunakan fungsi pop().

Contoh: