MODUL PRAKTIKUM
PENGANTAR PEMROGRAMAN

Program Studi Sistem Informasi
Departemen Matematika
Fakultas Matematika dan Pengetahuan Alam
Universitas Hasanuddin
Makassar
2022

Kata Pengantar

Dengan menyebut nama Allah SWT yang Maha Pengasih lagi Maha Penyayang, Kami panjatkan puja dan puji syukur atas kehadirat-Nya, yang telah melimpahkan rahmat, hidayah, dan inayah-Nya kepada kami, sehingga kami dapat menyelesaikan modul sistem basis data.

Modul ini telah kami susun dengan maksimal dan mendapatkan bantuan dari berbagai pihak. Untuk itu kami menyampaikan banyak terima kasih kepada semua pihak yang telah berkontribusi dalam pembuatan modul ini.

Terlepas dari semua itu, Kami menyadari sepenuhnya bahwa masih ada kekurangan baik dari segi susunan kalimat maupun tata bahasanya. Oleh karena itu dengan tangan terbuka kami menerima segala saran dan kritik dari pembaca agar kedepannya modul ini dapat semakin baik.

Akhir kata kami berharap semoga modul sistem basis data ini dapat memberikan manfaat maupun inspirasi bagi pembaca.

Makassar, September 2022

Tim Penyusun

Bab I

Intro to Python

Python sebagai bahasa pemrograman yang populer dan komprehensif dengan menggabungkan kapabilitas, sintaksis kode yang jelas serta dilengkapi pustaka standar yang mempunyai fungsionalitas sangat besar. Python termasuk dari jajaran bahasa pemrograman tingkat tinggi seperti bahasa pemrograman C, C++, Java, Perl dan Pascal.

A. Keywords

Keywords adalah kata yang dicadangkan oleh Python sehingga tidak boleh digunakan sebagai nama variabel, fungsi atau identifier lainnya. Keywords sendiri digunakan untuk mendefinisikan sintaks dan struktur dari bahasa Python. Keyword dalam Python merupakan case sensitive.

Ada 33 keyword dalam Python 3.7. Semua keywords selain True, False, dan None ditulis dalam lowercase dan harus ditulis apa adanya. Berikut adalah keywords dalam Python.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

B. Identifiers

Identifiers merupakan nama yang digunakan untuk mengidentifikasi class, function, variable, dll. Identifiers berguna untuk membedakan suatu entitas dengan entitas lainnya.

Aturan dalam menulis identifiers :

1. Identifiers dapat berupa kombinasi dari huruf dalam lowercase (a-z) atau uppercase (A-Z) atau digit (0-9) atau underscore (_). Contoh : myClass, var_1, this_is_a_long_variable
2. Identifiers tidak boleh diawali dengan digit.

```
1variable # Penulisan Salah
variable1 # Penulisan Benar
```

3. Keywords tidak dapat digunakan sebagai identifiers.

```
True = 1
```

Output:

```
File "d:\Code\tes.py", line 1
  True = 1
       ^
SyntaxError: invalid syntax
```

4. Tidak diperbolehkan menggunakan symbol special seperti !, @, #, $, %, dll.

```
x$ = 1
```

Output:

```
File "d:\Code\tes.py", line 1
  x$ = 1
      ^
SyntaxError: invalid syntax
```

5. Sebuah identifier dapat memiliki panjang berapapun.
6. Identifier Class dimulai dengan huruf uppercase sedangkan semua identifiers dimulai dengan huruf lowercase.
7. Identifier yang dimulai dengan underscore (_<name>) menandakan identifier tersebut merupakan identifier private.
8. Identifier yang dimulai dengan dua buah underscore (__<name>) menandakan identifier tersebut merupakan identifier yang sangat private.
9. Jika identifier tersebut juga diakhiri dengan dua buah underscore (__<name>__), identifier tersebut merupakan nama yang telah ditetapkan Python.

Hal yang perlu diingat!

· Python merupakan bahasa pemrograman yang case-sensitive. Sehingga, Variable dan variabel tidak sama.

C. Statements

Statements adalah instruksi atau pernyataan yang diberikan untuk dieksekusi oleh mesin. Penulisan statements dalam Python tidak diakhiri dengan tanda titik koma (;). Contohnya sebagai berikut:

```
x = 1
y = x
z = x + y
```

Dalam Python, akhir dari statement ditandai dengan baris baru. Tapi, kita dapat memperpanjang sebuah statement lebih dari beberapa baris secara eksplisit dengan menggunakan karakter forward slash (\). Contohnya :

```
x = 1 + 2 + 3 + \
  4 + 5 + 6 + \
  7 + 8 + 9
```

Multi-line statements juga terdapat dalam tanda kurung (), kurung siku [], kurung kurawal {}. Sebagai contoh, kita dapat mengimplementasikan contoh di atas menjadi :

```
x = (1 + 2 + 3 +
  4 + 5 + 6 +
  7 + 8 + 9)
colors = ['red',
      'blue',
      'green']
```

Kita juga dapat menyingkat penulisan statements menjadi satu baris menggunakan tanda titik koma ;.

```
a = 1; b = 2; c = 3
```

D. Indentation

Sebagian besar Bahasa pemrograman seperti C, C++, dan Java menggunakan kurung kurawal {} untuk mendefinisikan sebuah blok kode sedangkan Python menggunakan indentasi. Indentation sendiri adalah penulisan yang menjorok masuk ke dalam dari sebuah kode.

Sebuah blok kode (body dari sebuah function, loop, etc) dalam Python dimulai dengan indentasi dan diakhiri dengan baris yang tidak diindentasi. Jumlah spasi dari indentasi itu bebas, tetapi jumlah spasinya harus konsisten. Biasanya, 4 spasi digunakan sebagai indentasi dan lebih dipilih daripada tab. Sebagai contoh:

- Penulisan Benar

```
for i in range(1, 20):
   if i == 3:
      print("it's three")
      Break
```

- Penulisan Salah

```
for i in range(1, 20):
if i == 3:
print("it's three")
Break
```

Penggunaan dari indentasi dalam Python membuat kode terlihat rapi dan bersih sehingga menghasilkan sebuah kode yang terlihat mirip dan konsisten. Indentasi membuat kode tersebut menjadi lebih mudah dibaca. Sebagai contoh:

```
if True:
   print('Yes')
   x = 10
```

akan lebih mudah dibaca daripada,

```
if True:print('Yes'); x = 10
```

Indentasi yang salah akan menghasilkan error **IndentationError**.

E. Comments

Comments sangatlah penting dalam penulisan program. Comments membantu mendeskripsikan isi dari kode tersebut sehingga orang lain tidak sulit dalam memahami kode yang kita tulis. Penulisan comment dalam Python terbagi menjadi:

- Single-line comment

```
#This is a single-line comment
```

- Multi-line comment

```
"""
This is
a
Multi-line comments
"""
```

F. Variables & Constant

Variabel merupakan representasi dari alamat memori yang digunakan untuk menyimpan nilai dari data. Sintaks dari penulisan variabel adalah **name = value**. Contohnya,

```
x = 10
```

Di sini, kita telah membuat sebuah variabel bernama **x** dan telah memberinya value **10**. Variabel dapat kita anggap sebagai tas untuk menyimpan buku di dalamnya dan buku itu dapat diganti kapan saja. Hal ini berarti sebuah value dari variabel dapat diubah-ubah. Sebagai contoh:

```
x = 10
print(x)
x = 10.5
print(x)
```

Output:

```
10
10.5
```

Kita juga dapat memberikan beberapa nilai ke beberapa variabel sekaligus. Contoh:

```
x, y, z = 1, 3.2, "System Information"
print(x)
print(y)
print(z)
```

Output:

```
1
3.2
System Information
```

Jika kita ingin menetapkan value yang sama ke banyak variabel sekaligus, kita dapat melakukannya seperti:

```
x = y = z = "System Information"
print(x)
print(y)
print(z)
```

Output:

```
System Information
System Information
System Information
```

Constant merupakan sebuah tipe variabel yang valuenya tidak dapat diubah. Constant dapat kita anggap sebagai sebuah tas untuk menyimpan sebuah buku yang isinya tidak dapat diubah lagi. Contoh dari constant adalah

```
PI = 3.14
GRAVITY = 9.8
```

Aturan dan Ketentuan dalam penamaan variabel dan constant:

1. Nama constant dan variabel harus memiliki kombinasi huruf kecil (a-z) atau huruf besar (A-Z) atau angka (0-9) atau garis bawah (_)

   ```
   snake_case
   MACRO_CASE
   camelCase
   CapWords
   ```

2. Buat nama yang masuk akal. Misalnya length lebih masuk akal daripada l.
3. Jika ingin menulis nama variabel yang lebih dari dua kata, gunakan garis bawah untuk memisahkannya.

   ```
   car_name
   this_is_a_variable
   ```

4. Gunakan huruf kapital untuk mendeklarasikan sebuah constant.

   ```
   PI
   G
   MASS
   SPEED_OF_LIGHT
   TEMP
   ```

5. Jangan gunakan simbol spesial seperti !, @, #, $, %, etc.
6. Jangan memulai nama variabel dengan angka.

G. Data Types

Data types adalah klasifikasi atau kategorisasi item data yang mewakili jenis nilai yang memberi tahu operasi apa yang dapat dilakukan pada data tertentu. Karena semuanya adalah objek dalam Python, tipe data sebenarnya adalah sebuah class dan variable adalah instance (object) dari class ini.

Ada berbagai macam tipe data di Python yang sering digunakan, sebagai berikut:

1. **Python Numbers**

   Dalam Python, tipe data numerik mewakili data yang memiliki value numerik. Nilai numerik dapat berupa Integer, Float, dan bilangan Complex. Nilai – nilai ini didefinisikan sebagai kelas int, float, dan complex dalam Python.

   a. Integer – Nilai ini diwakili oleh kelas int. Integer berisi bilangan bulat positif atau negative (tanpa pecahan atau decimal).
   b. Float – Nilai ini diwakili oleh kelas float. Float adalah bilangan real dengan representasi floating point atau ditentukan oleh titik decimal. Secara opsional, karakter e atau E yang diikuti dengan bilangan bulat positif atau negative dapat ditambahkan untuk menentukan notasi ilmiah. Nilai float sendiri hanya akurat hingga 15 angka decimal.
   c. Complex number. Bilangan kompleks diwakili oleh kelas complex. Ini dispesifikasikan sebagai (bagian real) _ (bagian imajiner). Misalnya 2 + 3j.

   Kita dapat menggunakan fungsi type() untuk mengetahui tipe dari tipe data tersebut.

```
x = 10
print("Type of x: ", type(x))

y = 10.2
print("Type of y: ", type(y))

z = 3 + 5j
print("Type of z: ", type(z))
```

   Output:

```
Type of x:  <class 'int'>
Type of y:  <class 'float'>
Type of z:  <class 'complex'
```

2. **Python Strings**

   Dalam Python, String adalah array byte yang mewakili karakter Unicode. String adalah kumpulan dari satu atau lebih karakter yang ditulis dalam tanda kutip tunggal **‘<text>’**, tanda kutip ganda **“<text>”**, tanda kutip

tiga ‘’’<text>’’’ atau “”””<text>”””””. Dalam Python, tidak ada tipe data **char** sehingga char adalah string dengan panjang satu. String sendiri diwakili oleh class str.

```
s = 'This is a single quotes string'
print(s)
s = "This is a double quotes string"
print(s)
# String dengan tanda kutip tiga dapat
# membuat multi-line string
s = '''This is a triple
   quotes string'''
print(s)
```

Output:

```
This is a single quotes string
This is a double quotes string
This is a triple
quotes string
```

Karena string merupakan sebuah array, maka operator slicing **[ ]** dapat digunakan untuk mengakses karakter dalam string.

```
s = "Hello World"
print(s[6])
print(s[6:11])
```

Output:

```
W
World
```

3. **Python List**

List adalah sebuah urutan item yang berurutan. List merupakan salat satu tipe data yang paling sering digunakan dalam Python dan sangat fleksibel yang berarti semua item dalam list tidak harus bertipe sama. Untuk mendeklarasi sebuah list

```
a = [1, 2.2, 'python']
```

4. **Python Tuple**

Tuple sendiri mirip dengan dengan list tetapi item dalam tuple tidak dapat diubah. Tuple setelah dibuat tidak dapat dimodifikasi. Ini dikarenakan

tuple digunakan untuk melindungi data dan biasanya lebih cepat daripada list yang dapat berubah secara dinamis. Tuple sendiri dideklarasikan menggunakan tanda kurung **(<item>, <item>, …)** dan itemnya dipisah menggunakan tanda koma.

```
a = (1, 2.2, 'python')
```

5. **Python Set**

Set adalah sebuah koleksi dari berbagai item unik. Set didefinisikan oleh item yang dipisah oleh koma di dalam sebuah kurung kurawal **{<item>, <item>, …}**. Item dari sebuah set tidak terurut. Karena set itu memiliki value unik maka set akan mengeliminasi value yang duplikat.

```
a = {1, 2.2, 'python'}
```

6. **Python Dictionary**

Dictionary adalah sebuah koleksi tak terurut dari pasangan key-value. Dictionary sering digunakan ketika berhadapan dengan data yang besar karena adanya key-value sehingga pengambilan data lebih optimal. Setiap pasangan key-value dari dictionary dipisahkan oleh tanda titik dua **:**, di mana setiap key dipisahkan oleh koma dan setiap key tidak boleh sama (unik). Dalam Python, dictionary dibuat dengan menaruh kumpulan pasangan key-value tersebut ke dalam sebuah kurung kurawal **{key: value, key: value, …}**. Key dari dictionary ialah case sensitive, yang berarti jika sebuah key memiliki nama yang sama namun penulisannya berbeda maka akan dianggap sebagai key yang berbeda.

```
a = {'python': 1, 'Python': 1}
```

7. **Python Boolean**

Boolean adalah tipe data yang dapat menampung dua nilai, yaitu **True** dan **False**, operasi dengan *Logical Operator* juga akan menghasilkan nilai boolean, sehingga tipe data ini biasa digunakan dalam penyeleksian kondisi dan perulangan. Nilai **True** akan mengembalikan nilai **1** dan **False** akan mengembalikan nilai **0**.

```
is_raining = True
is_walking = False
x = (1 == True)
```

```
y = (1 == False)
a = True + 4
b = False + 10
```

Selain tipe data di atas, berikut adalah tipe data lengkap dari Python:

- **Numeric data types**: *int, float, complex*
- **String data types**: *str*
- **Sequence types**: *list, tuple, range*
- **Binary types**: *bytes, bytearray, memoryview*
- **Mapping data type**: *dict*
- **Boolean type**: *bool*
- **Set data types**: *set, frozenset*

H. Conversion of Data Types

Sebuah proses pengubahan nilai dari suatu tipe data ke tipe data yang lainnya disebut konversi tipe data. Python sendiri memiliki 2 jenis konversi data:

- **Implicit Type Conversion**

  Dalam konversi implisit, Python akan secara otomatis mengubah tipe data dari nilai tersebut. Proses ini tidak membutuhkan campur tangan dari user. Sebagai contoh:

```
num_int = 345
num_flo = 3.45

new = num_int + num_flo

print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))

print("Value of new:",new)
print("datatype of new:",type(new))
```

  Output:

```
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of new: 348.45
datatype of new: <class 'float'>
```

Dari program di atas dapat dilihat bahwa kita akan melakukan operasi penjumlahan dari variabel integer **num_int** dan float **num_flo**. Hasil dari penjumlahan tersebut akan menghasilkan sebuah value yang bertipe data **float**, hal ini terjadi karena Python akan selalu mengkonversi tipe data kecil ke tipe data besar untuk menghindari adanya kehilangan data.

- **Explicit Type Conversion**

  Dalam konversi eksplisit, user mengkonversi tipe data dari sebuah objek ke tipe data yang dibutuhkan dengan bantuan fungsi konversi tipe seperti **int()**, **float()**, **str()**, dll. Proses konversi tipe ini biasa disebut sebagai *typecasting*. Syntax : **<required_datatypes>(expression)**.

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))

num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str

print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Output:

```
Data type of num_int: <class 'int'>
Data type of num_str before Type Casting: <class 'str'>
Data type of num_str after Type Casting: <class 'int'>
Sum of num_int and num_str: 579
Data type of the sum: <class 'int'>
```

Dari program di atas dapat dilihat bahwa kita akan melakukan operasi penjumlahan dari variabel integer **num_int** dan string **num_str**. Karena akan terjadi error bila tipe data integer dan string dijumlahkan langsung, maka perlu dilakukan konversi tipe data string ke integer dengan menggunakan fungsi **int()**. Setelah dilakukan konversi maka kedua variabel tersebut dapat dijumlahkan dan menghasilkan sebuah value yang bertipe data **integer**.

I. Input

Untuk menambah fleksibilitas dalam program, kita mungkin mau mengambil input dari user. Python menyediakan fungsi input() untuk mengambil inputan dari user lalu menyimpannya dalam sebuah variabel. Syntax dari input() adalah:

```
input([prompt])
```

dimana prompt adalah sebuah string yang ingin ditampilkan secara opsional. Fungsi dari input() akan mengembalikan sebuah value string.

```
name = input("Input a name : ")
print(name)
```

Output:

```
Input a name : Sistem Informasi
Sistem Informasi
```

J. Operators

Operator adalah sebuah symbol special dalam Python yang berguna untuk menjalankan komputasi aritmatika atau logika. Value yang dioperasikan oleh operator disebut operand.

Dalam Python ada beberapa operator, sebagai berikut:

1. Arithmetic Operators

Operator aritmatik digunakan untuk menjalankan operasi matematika seperti penjumlahan, pengurangan, perkalian, dll.

Operator	Arti	Contoh
+	Menjumlahkan dua operand atau unary plus	x + y + 2
-	Mengurangi operand kanan dari kiri atau unary minus	x – y -- 2
*	Mengalikan operand kiri dengan kanan	x * y
/	Membagikan operand kiri dengan kanan (akan menghasilkan float)	x / y
%	Modulus – hasil bagi dari operand kiri dengan kanan	x % y

//	Floor Division – pembagian yang hasilnya menjadi bilangan bulat disesuaikan ke kiri pada garis bilangan	x // y
**	Pangkat – operand kiri dipangkatkan oleh operand kanan	x ** y

2. Comparison (Relational) Operators

Operator komparasi digunakan untuk membandingkan value dan akan mengembalikan nilai **True** atau **False** berdasarkan kondisinya.

Operator	Arti	Contoh
>	Lebih besar dari – True jika operand kiri lebih besar dari kanan	x > y
<	Kurang dari – True jika operand kiri lebih kecil dari kanan	x < y
==	Sama dengan – True jika kedua operand bernilai sama	x == y
!=	Tidak sama dengan – True jika kedua operand tidak bernilai sama	x != y
>=	Lebih besar atau sama dengan – True jika operand kiri lebih besar atau sama dengan kanan	x >= y
<=	Lebih kecil atau sama dengan – True jika operand kiri lebih kecil atau sama dengan kanan	x <= y

3. Logical (Boolean) Operators

Operator	Arti	Contoh
and	True jika kedua operand bernilai True	x and y
or	True jika kedua atau salah satu operand bernilai True	x or y
not	True jika operand bernilai False	not x

4. **Bitwise Operators**

Operator bitwise bertindak pada operand seolah-olah mereka adalah string digit binary.

Pada tabel di bawah : misalkan x = 10 (0000 1010) dan y = 4 (0000 0100)

Operator	Arti	Contoh
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise Or	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

5. **Assignment Operators**

Operator assignment digunakan dalam Python untuk menetapkan nilai ke variable.

Operator	Arti	Sama Dengan
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5

>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

6. Identity Operator

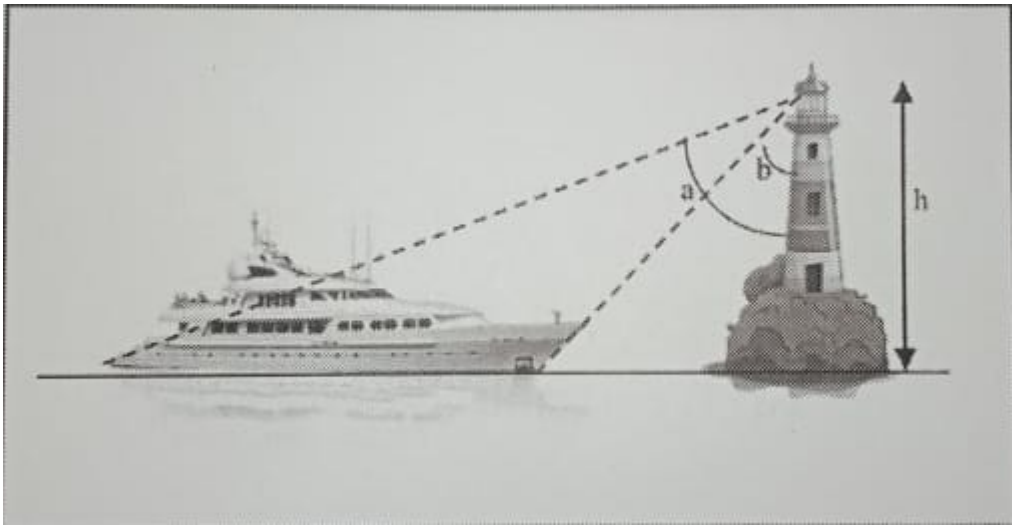
Operator	Arti	Contoh
is	True jika kedua operand identik	x is True
is not	True jika operand tidak identik	x is not True

7. Membership Operator

Operator	Arti	Contoh
in	True jika value/variable ditemukan dalam collections	x in True
not in	True jika value/variable tidak ditemukan dalam collection	x not in True

TUGAS PRAKTIKUM

1. Sebuah kapal sedang berlabuh dalam posisi menghadap ke menara (lihat pada gambar). Seorang pengamat (berada di puncak menara) ingin menghitung panjang kapal tersebut dengan mengetahui ketinggian menara (**h**), sudut elevasi pengamat terhadap ujung depan kapal (**b**) dan sudut elevasi pengamat terhadap ujung belakang kapal (**a**). Buatlah program untuk menghitung panjang kapal tersebut.



Format Input

Menerima tiga inputan **h**, **a**, dan **b** secara berurutan yang menyatakan ketinggian menara dalam satuan meter, sudut elevasi terhadap ujung depan kapal dan sudut elevasi terhadap ujung belakang kapal (90 > a > b).

Format Output

Menampilkan panjang kapal dengan tipe data *float* (satu angka dibelakang koma) dalam satuan meter.

Contoh Input 1 :

```
100
60
45
```

Contoh Output 1 :

```
73.2 m
```

Contoh Input 2 :

```
120
87
50
```

Contoh Output 2 :

```
2146.7 m
```

2. Buatlah program yang merubah detik ke dalam format jam:menit:detik

   Format Input

   Menerima satu inputan dengan tipe data Integer berupa detik

   Format Output

   Menampilkan hasil konversi dari detik ke format jam:menit:detik

   Contoh Input 1 :

   ```
   140153
   ```

   Contoh Output 1 :

   ```
   38:55:53
   ```

   Contoh Input 2 :

   ```
   270320
   ```

   Contoh Output 2 :

   ```
   75:05:20
   ```

Bab II

Conditional Statement

Conditional statement (yang dalam bahasa indonesia disebut pernyataan bersyarat atau pengkondisian) adalah fitur dari bahasa pemrograman yang melakukan perhitungan atau tindakan yang berbeda tergantung pada kondisi yang ditentukan programmer. Pengkondisian membuat program memiliki cabang yang masing-masing akan tereksekusi jika persyaratannya terpenuhi.

Pengkondisian biasanya dalam bentuk *if statement*, yaitu salah satu fitur utama dari bahasa pemrograman, tidak terkecuali Python. Hampir tidak ada bahasa pemrograman yang tidak memiliki *if statement* dan hampir tidak ada cara untuk memprogram tanpa cabang dalam aliran kode (setidaknya jika kode tersebut perlu memecahkan masalah yang kompleks).

Bahasa pemrograman seperti C, C++, dan Java setidaknya memiliki fitur conditional statement seperti *if statement* dan *switch case*, namun dalam bahasa python switch case baru di adaptasi pada versi 3.10 keatas yang dikenal sebagai "*structural pattern matching*" atau "*match case"*. Artinya, pada versi sebelumnya pengkondisian hanya dapat dilakukan menggunakan *if statement*. Ekspresi terdiri dari satu atau beberapa operator perbandingan dan operator logika yang akan menghasilkan nilai True (benar) atau False (salah).

Tabel Operator Perbandingan

Operator	Arti	Contoh	Hasil
>	Lebih besar dari – True jika operand kiri lebih besar dari kanan	5 > 6	False
<	Kurang dari – True jika operand kiri lebih kecil dari kanan	5 < 6	True
==	Sama dengan – True jika kedua operand bernilai sama	5 == 5	True
!=	Tidak sama dengan – True jika kedua operand tidak bernilai sama	5 != 5	False
>=	Lebih besar atau sama dengan – True jika operand kiri lebih besar atau sama dengan kanan	5 >= 3	True

<=	Lebih kecil atau sama dengan – True jika operand kiri lebih kecil atau sama dengan kanan	5 <= 5	True

Tabel Operator Logika

Operator	Arti	Contoh	Hasil
and	True jika kedua operand bernilai True	True and True	True
or	True jika kedua atau salah satu operand bernilai True	True or False	True
not	True jika operand bernilai False	not False	True

A. If-else Statement

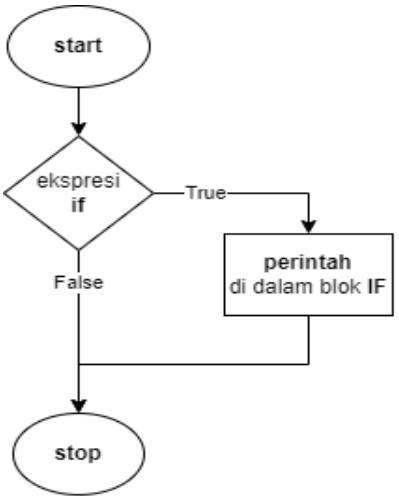
1. If statement

If statement adalah jenis pengkondisian yang paling mendasar, di mana kode dieksekusi apabila ekspresi terpenuhi atau bernilai *True* (benar). Pernyataan dari *if statement harus* memiliki *indent* minimal sepanjang satu spasi di awal tiap baris kode. Suatu pernyataan dapat berupa satu baris atau satu blok kode.

```
if ekspresi:

  # Pernyataan/statement(dieksekusi jika ekspresi
  # bernilai True)
```

Alur Program :



Contoh 1

```
angka = -5

if angka > 0:

    print(angka, "adalah bilangan positif")

print("pernyataan ini bernilai benar!")
```

Output:

```
> 5 adalah bilangan positif.
> pernyataan ini bernilai True!.
```

Contoh 2

```
angka = 5

if angka < 0:

    print(angka, "adalah bilangan negatif")

print("pernyataan ini bernilai False!")
```

Output:

```
> pernyataan ini bernilai False!.
```

Contoh 1.3

```
angka = -2

if angka > 0:

    print(angka, "adalah bilangan positif")

if angka < 0:

    print(angka, "adalah bilangan negatif")

print("pernyataan ini bernilai benar!")
```

Output:

```
> -2 adalah bilangan negatif.
```

2. if-else statement

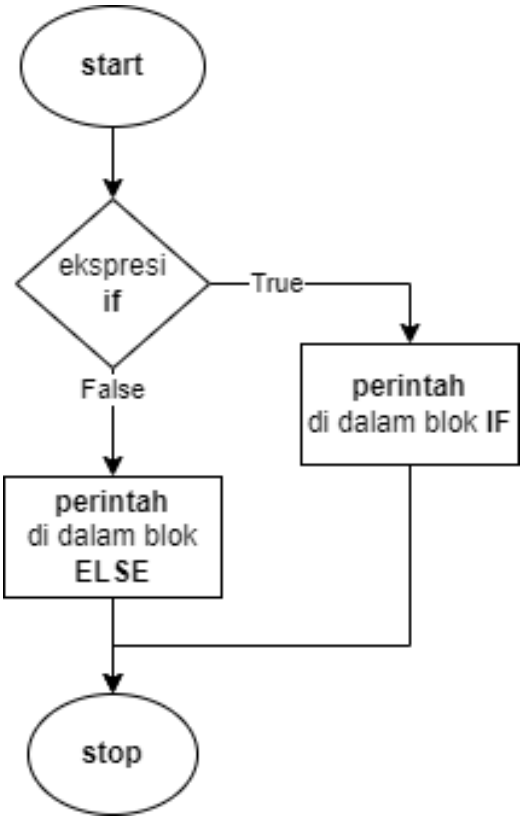
Pernyataan “else” digunakan ketika bagian benar dan salah dari kondisi tertentu ditentukan untuk dieksekusi. Ketika kondisinya benar, pernyataan di dalam blok if dieksekusi; jika kondisinya salah, program pada blok else akan dieksekusi.

Pernyataan if…else dalam Python memiliki sintaks berikut:

```
if ekspresi:

   # pernyataan/statement(dieksekusi jika
   # ekspresi bernilai True)
 else:
   # pernyataan/statement(dieksekusi jika
   # ekspresi bernilai False
```

Alur program:



Contoh 1

```
angka = 5

if angka >= 0:

    print("bilangan positif atau nol")

else:

     print(angka, “adalah bilangan negatif")
```

Output:

```
> bilangan positif atau nol
```

Contoh 2.2

```
x = 5

y = 20

if x == y:

    print("x dan y bernilai sama")

else:

     print(“x tidak sama dengan y")
```

Output:

```
> x tidak sama dengan y
```

3. If-elif-else statement

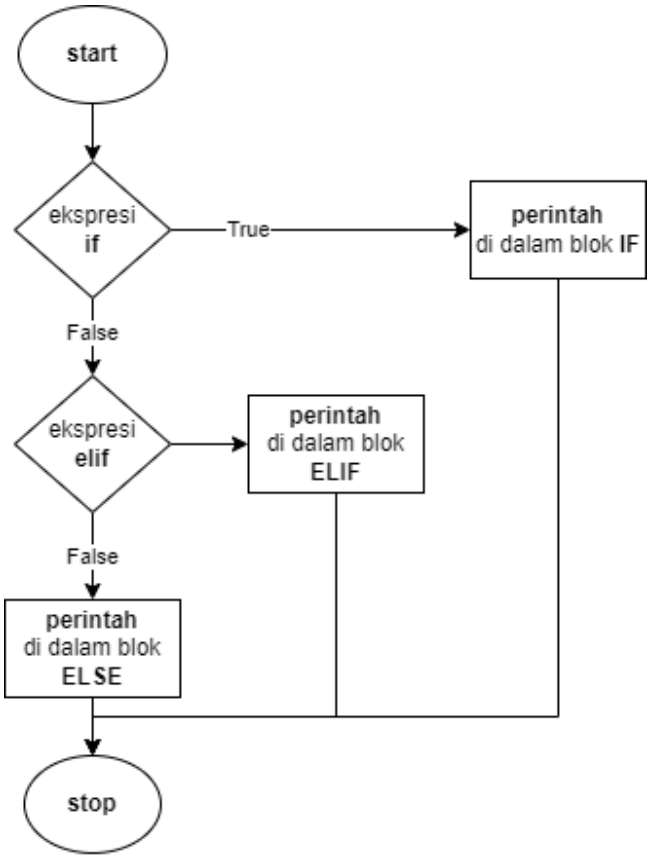
Pernyataan elif memungkinkan Anda untuk memeriksa beberapa ekspresi dan mengeksekusi blok kode segera setelah salah satu kondisi mengevaluasi ke True. Dalam hal ini, kondisi if dievaluasi terlebih dahulu. Jika salah, pernyataan elif akan dieksekusi, jika itu juga salah, pernyataan else akan dieksekusi.

Pernyataan If…Elif..else dalam Python memiliki sintaks berikut:

```
if ekspresi:

    #dieksekusi jika ekspresi bernilai True#
elif ekspresi:
    #dieksekusi jika ekspresi (elif) bernilai True
else:
    #dieksekusi jika ekspresi bernilai False
```

Alur program:



Contoh 3

```
angka = 0

if angka > 0:
```

```
    print(angka, "adalah bilangan positif")
elif angka == 0:
    print(angka, "adalah bilangan nol")
else:
     print(angka, “adalah bilangan negatif")
```

Output:

```
> 0 adalah bilangan nol
```

Contoh 2

```
angka = 0
if angka > 0:
    print(angka, "adalah bilangan positif")
elif angka == 0:
    print(angka, "adalah bilangan nol")
else:
     print(angka, “adalah bilangan negatif")
```

Output:

```
> 0 adalah bilangan nol
```

4. **Nested IF Statement**

Pernyataan IF bersarang adalah pernyataan di mana pernyataan If terletak di dalam pernyataan If lainnya. Ini digunakan ketika variabel harus diproses lebih dari sekali. Pernyataan if, if-else, dan if…elif…else dapat digunakan dalam program.

Pernyataan if bersarang dalam Python memiliki sintaks berikut:

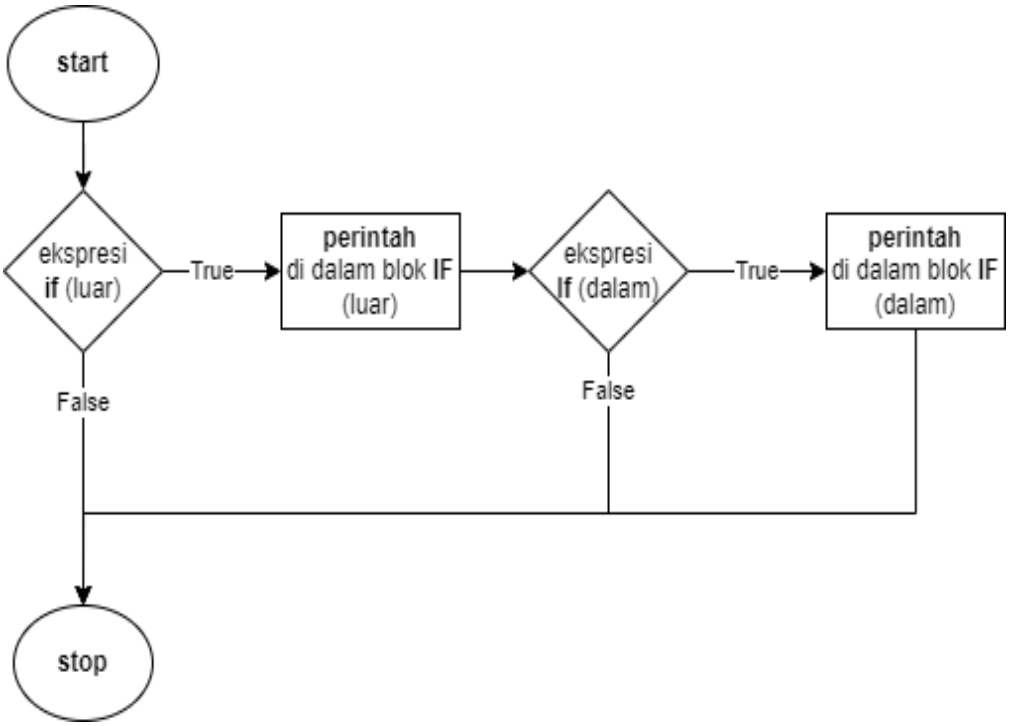
```
if ekspresi:

   #dieksekusi jika ekspresi bernilai True#

   if ekspresi:

      # dieksekusi jika ekspresi if (luar) dan
      # if (dalam) bernilai True
```

Alur program:



Contoh 1

```
angka = 0

if angka >= 0:
```

```
    print(angka, "adalah bilangan positif")
    if angka == 0:
        print(angka, "adalah bilangan nol")
else:
     print(angka, “adalah bilangan negatif")
```

Output:

```
> 0 adalah bilangan nol
```

Contoh 2

```
angka = 4
if angka >= 0:
    print(angka “adalah bilangan positif)
    if (angka % 2) == 0:
        print(angka, "adalah bilangan genap
            positif")
else:
     print(angka, “adalah bilangan negatif")
```

Output:

```
> 0 adalah bilangan nol
```

5. **Shorthand if statement**

Shorthand if statement digunakan ketika hanya satu statement yang perlu dieksekusi di dalam blok if. Pernyataan ini dapat disertakan di baris yang sama dengan pernyataan If. Pernyataan

Short Hand if dalam Python memiliki sintaks berikut:

```
if kondisi: statement
```

Contoh program:

```
a = 123

if a > 100: print(a, “lebih besar dari seratus”)
```

Output:

```
> 123 lebih besar dari 100
```

6. **ShortHand if-else statement**

Shorthand if-else digunakan untuk menyebutkan pernyataan If-else dalam satu baris di mana hanya ada satu pernyataan untuk dieksekusi di blok if dan else.

Short Hand if-else dalam Python memiliki sintaks berikut:

```
statement (True) if kondisi else statement (False)
```

Contoh program:

```
a = 123

print(a, “ besar”)if a > 100 else print(a, “ kecil”)
```

Output:

```
> 123 besar
```

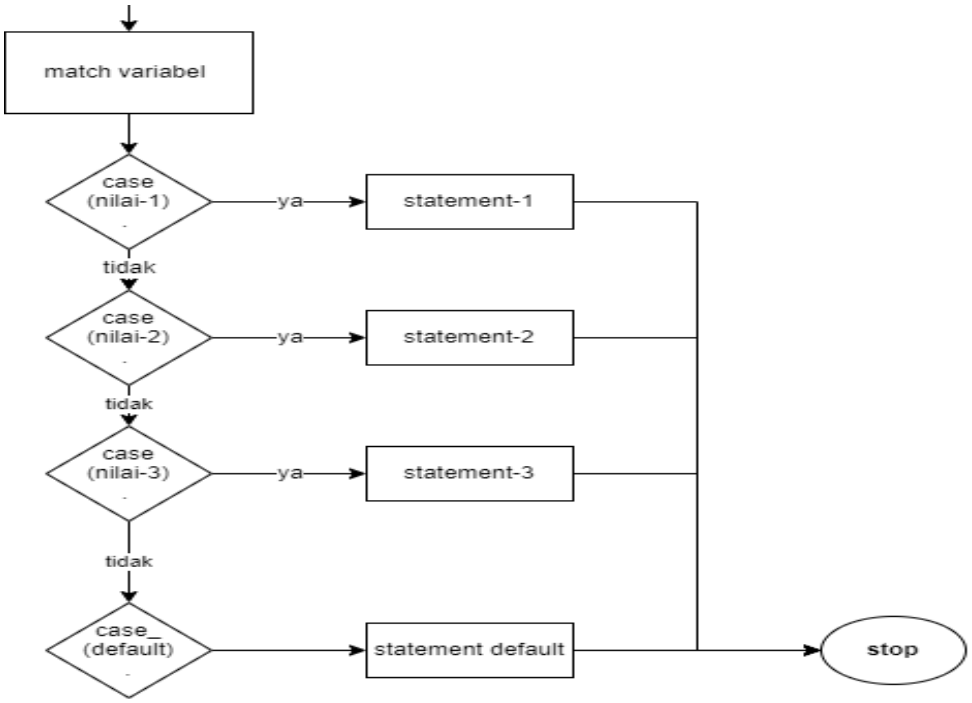
B. Match Case

Pernyataan switch mengevaluasi ekspresi, mencocokkan nilai ekspresi terhadap serangkaian klausa kasus, dan mengeksekusi pernyataan setelah klausa kasus pertama dengan nilai yang cocok, hingga pernyataan break ditemukan. Klausa default dari pernyataan switch akan dilompati jika tidak ada kasus yang cocok dengan nilai ekspresi.

Short Hand if dalam Python memiliki sintaks berikut:

```
match variabel:
    case nilai-1:
        statement-1
    case nilai-2:
        statement-2
    case nilai-…:
        statement-…
    case _:
        statement-default (jika tidak ada case yang sesuai)
```

Alur program:



Contoh program:

```
bahasa = "python"
match bahasa:
    case "JavaScript":
        print("kamu akan menjadi web developer.")

    case "Python":
        print("kamu akan menjadi Data Scientist")

    case "PHP":
        print("kamu akan menjadi backend developer")

    case "Solidity":
        print("kamu akan menjadi Blockchain developer")

    case "Java":
        print("kamu akan menjadi mobile app developer")

    case _:

        print("bahasa tidak penting, yang penting adalah
              mampu untuk menyelesaikan masalah.")
```

Output:

```
> kamu akan menjadi Data Scientist
```

TUGAS PRAKTIKUM

1. Buatlah program yang dapat mengkonversi nilai dalam bentuk angka ke huruf dengan aturan:

nilai >= 90	A
nilai >= 80	B
nilai >= 70	C
nilai >= 60	D
nilai >= 40	E
nilai < 40	F

Format Input:

Program menerima inputan nilai dengan tipe data numerik

```
Nilai = 87
```

Format Output:

Cetak hasil dengan format: "{nilai dari bilangan terbesar} adalah nilai terbesar"

```
> nilai 87 = 'B'
```

2. Buatlah program untuk menginput informasi daya listrik dan menghitung total tagihan listrik berdasarkan aturan tarif berikut:

Golongan	Daya	Tarif/kWh
R1	900 VA	Rp 1.352
R1	1.300 VA	Rp 1.444,70
R1	2.200 VA	Rp 1.444,70
R2	3.500 VA-5.500 VA	Rp 1.699,53

R3	6.600 VA - ke atas	Rp 1.699,53
B2	6.600 VA-200 kVA	Rp 1.444,70
B3	Di atas 200 kVA	Rp 1.114,74
I3	200 kVA - ke atas	Rp 1.314,12
P1	6.600 VA-200 kVA	Rp 1.523,28

Format input:

Program menerima input “golongan”, “daya”, dan “pemakaian” sesuai dengan aturan yang disediakan.

```
golongan = B2

daya = 150000

Pemakaian = 8914
```

Format output:

Cetak hasil perhitungan dengan format: “jumlah tagihan anda : {tagihan}”. Jika data input tidak terdapat pada aturan diatas, cetak “data tidak valid”.

```
> jumlah tagihan anda : 12.878.055,8
```

3. Buatlah program untuk menentukan nilai terbesar dari tiga angka!

Format Input:

Program menerima tiga inputan bilangan yang masing -masing ditampung ke dalam variabel a,b, dan c.

```
a = 21

b = 13

c = 90
```

Format Output:

Cetak hasil dengan format: “{nilai dari bilangan terbesar} adalah nilai terbesar”

```
> 90 adalah nilai terbesar
```

Bab III

Looping

Looping atau perulangan merupakan cara yang digunakan untuk menjalankan perintah yang berulang untuk data berbentuk kelompok seperti list, tuple atau string. Dalam melakukan perintah looping biasanya digunakan perintah For Loop dan While Loop, serta penerapan pernyataan Break dan Continue sebagai pilihan tambahan dalam membuat perintah looping.

A. For Loop

Perintah “For” digunakan dalam melakukan perulangan pada data kelompok yang sudah diketahui jumlah iterasinya dan akan berhenti jika iterasinya telah dieksekusi semuanya.

Contoh:

```
 fruits = ["apel", "pisang", "mangga"]
for x in fruits:
 print("Ini adalah", x)
```

Output:

```
Ini adalah apel
Ini adalah pisang
Ini adalah mangga
```

B. While Loop

Perintah “While” digunakan dalam melakukan perulangan pada data kelompok yang tidak diketahui jumlah pasti iterasinya. Oleh karena itu, biasanya perintah while akan terus melakukan perulangan selama suatu kondisi masih terpenuhi. Selain itu, pada looping juga kita bisa tambahkan kondisi “else” seperti layaknya pada Conditional Statement pada bab sebelumnya.

Contoh:

```
n = int(input("Enter n: "))

sum = 0
i = 1

while i <= n:
   sum = sum + i
   i = i+1
else:   #Penambahan kondisi else optional
   print("The sum is", sum)
```

Output:

```
Enter n: 5
The sum is 15
```

C. Break dan Continue

Penggunaan “Break” dan “Continue” digunakan untuk menambahkan kondisi perintah pada looping sehingga dalam looping tersebut akan ada kondisi “If” yang ditambahkan. Break digunakan jika kita ingin menghentikan sebuah looping jika suatu kondisi telah terpenuhi, sedangkan continue digunakan untuk melewati iterasi sekarang dan melanjutkan langsung ke iterasi selanjutnya jika kondisinya terpenuhi.

1. Break Statement

```
fruits = ["apel", "pisang", "mangga"]
for x in fruits:
  if x == "pisang":
    Break
  print("Ini adalah", x)
```

Output:

```
Ini adalah apel
```

2. Continue Statement

Sama halnya pada "For Loop" statement di atas juga dapat digunakan pada "While Loop" dengan penulisan yang sama. Menambahkan conditional statement ke dalam looping yang dibuat.

```
fruits = ["apel", "pisang", "mangga"]
for x in fruits:
  if x == "pisang":
    continue
  print("Ini adalah", x)
```

Output:

```
Ini adalah apel
Ini adalah mangga
```

D. Nested Loop

Nested loop atau perulangan bersarang adalah sebuah perulangan yang di dalamnya terdapat perulangan yang lain. Baik itu perulangan While-For, While-While, For-For atau berbagai macam kombinasi Nested Loop.

Contoh:

```
adj = ["red", "tasty"]
fruits = ["apple", "banana"]

for x in adj: #Loop Pertama
  for y in fruits: #Loop Kedua
    print(x, y)
```

Output:

```
red apple
red banana
tasty apple
tasty banana
```

E. Try-Exception

Ketika terjadi kesalahan, atau exception, Python biasanya akan berhenti dan menghasilkan pesan kesalahan. Pengecualian ini dapat ditangani menggunakan pernyataan try:

Contoh 1 :

```
x = 3
y = 'Empat'
try:
    z = x + y
except:
    z = str(x) + y

print(z)
```

```
3Empat
```

Contoh 2:

```
x = 3
y = 'Empat'
try:
    z = x + y
except:
    z = y + str(x)
else:
    print('Tidak dapat menjumlahkan keduanya')

print(z)
```

```
Empat3
```

Contoh 3:

```
x = 3
y = 'Empat'
try:
    z = x + y
except:
    z = y + str(x)
finally:
    print('!!! salah satunya dilakukan conversi type data')

print(z)
```

```
!!! salah satunya dilakukan conversi type data
Empat3
```

TUGAS PRAKTIKUM

1. Buatlah program untuk mencetak angka seperti berikut

   Format Input:

   Input terdiri atas X dan Y (X < Y)

```
4
100
```

   Format Output:

   Cetak hasil secara horizontal sebanyak X, kemudian cetak semua nilai dari 1 hingga Y secara vertikal.

```
1 2 3 4
5 6 7 8
. . .
97 98 99 100
```

2. 1 hari sama dengan 24 jam. Nah bagaimana jika 1 hari disama dengankan dengan 360^o. Maka cetak nilai n (dalam derajat) ke dalam bentuk satuan jam yaitu HH:MM:SS. Dengan nilai 0^o sama dengan jam 6:00:00 dan seterusnya.

   Format Input:

   Inputan terdiri atas 1 bilangan float M ($0 \leq M < 360$) yang menunjukkan posisi matahari atau bulan. Inputan akan berhenti dengan end of file (EOF).

   Format Output:

   Cetak selamat pagi, Selamat Siang, Selamat Sore, Selamat Malam berdasarkan waktunya yang diikuti dengan waktu aslinya.

   Contoh Input 1:

```
0.9
```

   Contoh Output 1:

```
Selamat Pagi
06:03:36
```

   Contoh Input 2:

```
90.0
```

Contoh Output 2:

```
Selamat Siang
12:00:00
```

3. Buatlah program untuk menghitung kembalian dari suatu transaksi.

   Format Input:

   Inputan terdiri atas dua bilangan integer yaitu harga barang dan nilai uang yang dibayarkan.

```
580000
1000000
```

   Format Output:

   Cetak kembalian dari transaksi tersebut disertai dengan detail jumlahnya, dalam bentuk uang pecahan yang sesuai.

```
4 uang Rp. 100.000
0 uang Rp. 50.000
1 uang Rp. 20.000
0 uang Rp. 10.000
0 uang Rp. 5.000
0 uang Rp. 2.000
0 uang Rp. 1.000
```

Bab IV

Functions

Dalam konteks pemrograman, *function* atau fungsi didefinisikan sebagai sekumpulan baris perintah atau kode yang dikelompokkan menjadi satu kesatuan yang akan dieksekusi ketika dia dipanggil. Sebuah *function* bisa menerima parameter, bisa mengembalikan suatu nilai, dan bisa dipanggil berkali-kali secara independen. Tujuan dari *function* adalah untuk memecah program yang besar dan kompleks menjadi bagian-bagian kecil dengan tugasnya masing-masing.

A. Pembuatan Function

Dalam bahasa pemrograman Python, *function* dibuat dengan sintaks sebagai berikut:

```
def <function_name>(parameters):
  statements
```

Keterangan:

1. *def* adalah sintaks yang mendefinisikan bahwa blok kode program adalah sebuah *function*
2. <function_name> adalah nama dari *function*
3. (parameters) adalah nilai-nilai (satu atau lebih) yang dikirimkan ke dalam *function* yang akan dieksekusi di dalam *function body*
4. *statements* adalah kumpulan perintah yang akan dieksekusi ketika *function* dipanggil

Catatan:

Blok kode program di dalam Python didefinisikan dengan indentasi

Contoh Function:

```
def hello_world():
  print('Hello python! Hello World!'
```

B. Pemanggilan Function

Function dipanggil dengan menuliskan nama fungsinya diikuti tanda kurung () sebagai berikut:

```
hello_world()
```

Output:

```
Hello python! Hello World!
```

Function juga dapat digunakan atau dipanggil berkali-kali:

```
hello_world()

hello_world()

hello_world()
```

Output:

```
Hello python! Hello World!

Hello python! Hello World!

Hello python! Hello World!
```

C. Function dengan Parameter

Sebuah *function* dapat menerima satu atau lebih parameter atau argumen yang merupakan nilai yang dikirimkan ke dalam fungsi untuk dieksekusi dalam *function body*.

```
def welcome(name):

  print(f'Hello {name}, welcome!')
```

Keterangan:

name adalah sebuah parameter yang dikirimkan ke dalam fungsi untuk ikut dieksekusi pada perintah *print*()

```
welcome('Human')
```

```
welcome('Robot')

welcome('Alien')
```

Output:

```
Hello Human, welcome!

Hello Robot, welcome!

Hello Alien, welcome!
```

D. Function dengan Parameter Wajib

Parameter di dalam Python dapat bersifat wajib, artinya parameter tersebut harus diisi. Perhatikan contoh berikut:

```
def introduce(name, origin):

  print(f"I'm {name} from {origin}")
```

Parameter *name* dan *origin* adalah parameter wajib. Jika *function* tersebut dipanggil dengan kedua parameternya:

```
introduce('Human', 'Earth')
```

Output:

```
I'm Human from Earth
```

Jika *function* tersebut dipanggil hanya dengan salah satu parameternya:

```
introduce('Human')
```

Output:

```
Exception has occurred: TypeError

introduce() missing 1 required positional argument:
'origin'
```

Terjadi *exception* karena terdapat parameter tidak lengkap atau hilang.

E. Function dengan Parameter Opsional

Parameter di dalam Python juga dapat bersifat opsional, artinya parameter yang dapat tidak diisi karena memiliki nilai *default*. Perhatikan contoh berikut:

```
def introduce (name, origin='Somewhere'):
  print(f"I'm {name} from {origin}")
```

Parameter *name* adalah parameter wajib sedangkan *origin* adalah parameter opsional. Jika *function* tersebut dipanggil dengan parameter wajibnya:

```
introduce('Human')
```

Output:

```
I'm Human from Somewhere
```

Jika *function* tersebut dipanggil dengan parameter wajib dan parameter opsionalnya:

```
introduce('Alien', 'Space')
```

Output:

```
I'm Alien from Space
```

F. Function dengan Parameter Berurut dan Tidak Berurut

Sebuah *function* dapat memiliki parameter opsional yang lebih dari satu, artinya terdapat beberapa parameter yang dapat diisi ataupun tidak diisi karena memiliki nilai default. Perhatikan contoh berikut:

```
def introduce (name, origin='Somewhere', power='no
power'):
  print(f"I'm {name} from {origin}, I have {power}")
```

name adalah parameter wajib, sedangkan origin dan power adalah parameter opsional. Jika *function* tersebut dipanggil dengan parameternya secara lengkap dan berurut:

```
introduce('Human', 'Earth', 'a mind')
```

Output:

```
I’m Human from Earth, I have a mind
```

Jika *function* tersebut dipanggil dengan parameternya secara lengkap tapi tidak berurut:

```
introduce(‘Human’, ‘a mind’, ‘Earth’)
```

Output:

```
I’m Human from a mind, I have Earth
```

Jika *function* tersebut dipanggil dengan parameternya secara tidak lengkap tapi berurut:

```
introduce(‘Human’, ‘Earth’)
```

Output:

```
I’m Human from Earth, I have no power
```

Jika *function* tersebut dipanggil dengan parameternya secara tidak lengkap dan tidak berurut:

```
def introduce(‘Human’, ‘a mind’)
```

Output:

```
I’m Human from a mind, I have no power
```

Dari keempat program diatas diketahui bahwa yang menghasilkan output yang benar adalah function yang parameternya dipanggil secara berurutan. Sedangkan, *function* yang parameternya dipanggil secara tidak berurutan menghasilkan output yang salah.

Pemanggilan *function* dengan parameter yang tidak berurutan dapat dilakukan dengan mendefinisikan nama parameter beserta nilainya untuk menghasilkan output yang benar. Perhatikan contoh berikut:

```
introduce(name='Alien', power='a strength',
origin='Space')
```

Output:

```
I'm Alien from Space, I have a strength
```

G. Function yang Mengembalikan dan Tidak Mengembalikan Nilai

Ditinjau dari segi pengembalian nilai, *function* dapat dikategorikan menjadi 2 bagian:

1. Function yang Mengembalikan Nilai

Function yang mengembalikan nilai adalah jenis *function* yang jika dipanggil akan mengembalikan nilai sebagai hasil dari pemanggilan *function* tersebut. Nilai tersebut dapat digunakan untuk perintah lebih lanjut seperti menyimpan nilai dalam variabel, melakukan perhitungan, atau mencetak nilai, dsb.

Perhatikan contoh berikut:

```
def add2values(a, b):
  result = a + b
  return result
```

Dalam sebuah *function*, untuk mengembalikan nilai digunakan sintaks *return*. Jika *statement return* telah dieksekusi pada sebuah *function*, maka semua proses yang ada di dalam blok code function tersebut akan berhenti dieksekusi.

```
#Case01 (just call the function)
add2values(1, 2)
#Case02 (store the function in a variable)
a = add2values(1, 2)
#Case3 (print the function)
print("1 + 2 =", add2values(1, 2))
```

Output:

```
#Output01


#Output02


#Output03
1 + 2 = 3
```

2. Fungsi yang Tidak Mengembalikan Nilai

Fungsi yang tidak mengembalikan nilai adalah fungsi yang ketika dipanggil hanya mengeksekusi perintah pada *body function* tanpa mengembalikan nilai yang ditandai dengan tidak adanya *statements* return. Perhatikan contoh berikut:

```
def add2values(a, b):
  result = a + b
  print(a, '+', b, '=', result)

add2values(1, 2)
```

Output:

```
1 + 2 = 3
```

H. Function Rekursif

Function rekursif adalah *function* yang memanggil dirinya sendiri dalam *function body*-nya yang menyebabkan efek perulangan. Perulangan ini bisa berhenti ketika kondisi tertentu tercapai, atau bisa juga bersifat tak terbatas, atau mungkin bahkan bisa menimbulkan error karena pemanggilan fungsi yang tak ada habisnya.

Function rekursif dengan perulangan terbatas;

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

  if(start <= end):
     show_intervals(start, end)

show_intervals(1, 10)
```

Output:

```
1
2
3
. . .
9
10
```

Function rekursif dengan perulangan tanpa batas;

```
def show_multiple(value):
   print(value)
   value = value + value
   show_multiple(value)
show_multiple(2)
```

Output:

```
2

4

6

8

10

. . .

. . .

. . .

RecursionError: maximum recursion depth exceeded while
calling a Python object
```

TUGAS PRAKTIKUM

1. Buatlah sebuah program yang menerima inputan sebuah bilangan bulat kemudian mengembalikan nilai yang menentukan bilangan tersebut termasuk bilangan prima atau tidak menggunakan *function*.

   Contoh Input:

   ```
   6079
   ```

   Contoh Output:

   ```
   Ini bilangan prima
   ```

2. Buatlah sebuah program yang menerima inputan sebuah bilangan bulat kemudian mengembalikan nilai faktorial dari bilangan tersebut menggunakan *rekursif function*.

   Contoh Input:

   ```
   8
   ```

   Contoh Output:

   ```
   40320
   ```

3. Buatlah sebuah program yang menerima sebuah bilangan yang menyatakan panjang sebuah list, dan sebuah list berupa bilangan bertipe *integer* kemudian mencetak list tersebut dengan bilangan yang sudah terurut dari kecil ke besar menggunakan *function*. (tanpa menggunakan *built-in function* pada Python seperti sort())

   Contoh Input:

   ```
   6
   7 4 2 1 3 2
   ```

   Contoh Output:

   ```
   1 2 2 3 4 7
   ```

4. Buatlah sebuah program untuk menghitung Faktor Persekutuan Terbesar (FPB) dari dua inputan angka *integer* dengan menggunakan *function* getFPB() yang menerima dua parameter integer dan menghasilkan output FPB dari dua bilangan tersebut.
   Contoh Input:

```
20
199
```

   Contoh Output:

```
FPB (20, 199) = 1
```

5. Baca nilai *integer* yang sesuai usia seseorang (dalam hari), kemudian cetak dalam tahun, bulan, dan hari, diikuti keterangan masing-masing ‘tahun’, ‘bulan’, dan ‘hari’ menggunakan *function* myDay().
   Catatan: hanya untuk memudahkan perhitungan, pertimbangkan seluruh tahun dengan 365 hari dan 30 hari setiap bulan. Dalam kasus pengujian tidak akan pernah ada situasi yang memungkinkan 12 bulan dan beberapa hari, seperti 360, 363 atau 364. Ini hanya latihan untuk tujuan pengujian penalaran matematika sederhana.
   Contoh Input 1:

```
400
```

   Contoh Output 1:

```
1 tahun
1 bulan
5 hari
```

   Contoh Input 2:

```
800
```

   Contoh Output 1:

```
2 tahun
2 bulan
10 hari
```

Bab V

Collections

Collections adalah kumpulan dari beberapa nilai baik itu angka, string, bahkan variable dalam satu grup. Collection terbagi menjadi tiga yaitu *List, Tuples, Dictionary, Set*.

A. List

List digunakan untuk menyimpan beberapa item dalam satu variabel. List adalah salah satu dari 4 tipe data bawaan dalam Python yang digunakan untuk menyimpan kumpulan data

1. Membuat List

Untuk membuat variabel berdata type list sangat mudah, hanya perlu membungkus beberapa data dengan kurung siku, dan memberi *koma(,)* di antara data data nya**.**

Contoh:

```
listExample = [1, 2, 3]
```

List memiliki size yang berupa jumlah data yang ada di dalam list kita bisa mendapatkan nilai size pada list menggunakan *built-in function len()*

2. Mengakses Data List

Nilai dalam *list* dapat diakses dengan memanggil indeks *list*, indeks pada list dimulai dari 0 (nol) untuk data pertama dan **Size List - 1** untuk nilai paling akhir.

Contoh:

- Mengambil data pertama

  ```
  listExample[0]
  ```

- Mengambil data terakhir

  ```
  listExample[len(listExample) - 1]
  ```

Nilai dalam list dapat diubah dengan cara menginisialisasi nilai list pada indeks yang ingin diubah

Contoh:

```
listExample[1] = 10
```

3. List Multi Dimensi

List multi-dimensi adalah list yang memuat list di dalamnya hal ini digunakan sebagai representasi matriks di dunia matematika

Contoh:

```
listExample = [[1, 2],[2, 3]]
```

Dari contoh diatas bisa kita melihat bahwa variabel **listExample** adalah matriks 2x2, dan kita bisa melihatnya seperti berikut

| 1, 2 |
| 2, 3 |

Untuk mengambil datanya kita bisa menggunakan:

```
print(listExample[0][1])
```

Untuk mengambil matriks 1,1

4. Built-In Function Pada List

List memiliki banyak *built-in function* beberapa yang penting diantaranya;

- **Append**

  Kita dapat menambahkan nilai pada list menggunakan fungsi *append*.

  ```
  listExample.append(4)
  ```

- **Extend**

  Untuk menambahkan list langsung ke dalam list.

  ```
  listExample.extend([5, 6, 7])
  ```

- **Remove**

  Untuk menghapus nilai dalam list.

  ```
  listExample.remove(10)
  ```

- **Pop**

  Untuk menghapus nilai pada indeks tertentu dan mereturn nilai di indeks tersebut.

  ```
  listExample.pop(1)
  ```

- **Insert**

  Untuk menambahkan nilai dalam list pada indeks tertentu

```
listExample.insert(1, 2)
```

B. Dictionary

Dictionary adalah kumpulan nilai kunci, yang digunakan untuk menyimpan nilai data key dan value, tidak seperti list yang hanya menyimpan satu nilai sebagai elemen.

1. Membuat Dictionary

Dalam Python, dictionary bisa dibuat dengan menempatkan urutan elemen dalam kurung kurawal, dipisahkan dengan 'koma'. Kamus menampung pasangan nilai, satu menjadi Kunci dan elemen pasangan terkait lainnya menjadi Kunci nilainya. Nilai dalam kamus dapat berupa tipe data apa pun dan dapat diduplikasi, sedangkan kunci tidak dapat diulang (memiliki kata kunci sama).

Contoh:

```
dictExample = {
 "name": "Fatwa",
 "age": 22,
 "hobby": "Playing Onet",
}
```

2. Melakukan Operasi Pada Dictionary

Untuk Mengambil data dictionary kita bisa melakukannya seperti pada list

```
dictExample["name"]
```

Untuk mengupdate data pada dictionary kita hanya perlu mengintansinya kembali.

```
dictExample["hobby"] = "Playing sudoku"
```

Begitu pula untuk menambah data pada dictionary kita hanya perlu menginstansiasinya.

```
dictExample["graduationYear"] = 2022
```

C. Tuple

Tuple digunakan untuk menyimpan beberapa item dalam satu variabel. Tuple adalah kumpulan item dan tidak dapat diubah

1. Membuat Tuple

Penulisan tuple di mulai dan ditutup dengan tanda kurung.

Contoh:

```
tupleExample = ("Coding", True, False, "Adakah")
```

2. **Manipulasi Tuple**

   Tuple kumpulan beberapa item dan tidak dapat diubah lagi. Tetapi kita bisa melakukan manipulasi tuple tersebut, dengan cara melakukan konversi tipe data tuple ke list, lalu data list yang kita ubah, setelah diubah, dilakukan kembali konversi data dari list ke tuple.

   Contoh:

```
tupleExample = ("Coding", True, False, "Adakah")
tupleToList = list(tupleExample)
tupleToList[0] = "Dari Tuple Ke List"
tupleExample = tuple(tupleToList)
```

D. Set

Set adalah kumpulan yang tidak berurutan, tidak dapat diubah, dan tidak terindeks dan tidak ada anggota yang terduplikat. Sama halnya dengan tuple, set juga bisa memiliki item yang berbeda tipe datanya serta juga dapat diketahui panjangnya atau banyaknya item di dalam set dengan menggunakan fungsi len().

1. **Membuat Set**

   Set diinisiasi dengan cara membungkus data menggunakan kurung kurawal

```
setExample = {"Tes", 1, 2, "Mantap"}
```

2. **Memanipulasi Set**

   Sama halnya dengan tuple, set juga kumpulan beberapa item dan tidak dapat diubah lagi. Tetapi kita bisa melakukan manipulasi set tersebut, dengan cara melakukan konversi tipe data set ke list, lalu data list yang kita ubah, setelah diubah, dilakukan kembali konversi data dari list ke set.

3. **Mengambil data dari set**

   Untuk mengambil data dari ***set*** kita bisa memanggil fungsi pop, yang akan menghapus dan mengembalikan data terakhir dari set, jika kita tidak mau menghapus datanya ketika ingin mengambil datanya maka, kita harus meng-iterasinya atau kita bisa mengubahnya ke dalam bentuk list, karena kita tidak bisa mengambil data dari set langsung dari indeksnya seperti list karena hal itu akan menghasilkan error

   Contoh:

SALAH

```
contohSet = {1, 2, 3, 4, 5}
print(contohSet[1])
```

BENAR

```
contohSet = {1, 2, 3, 4, 5}
print(list(contohSet)[1])
```

Atau untuk mengambil semua data dalam set bisa juga

```
contohSet = {1, 2, 3, 4, 5}
for i in contohSet: print(i)
```

4. **Built-In Function di Set**

- **Add**

  Untuk menambah data pada set

  ```
  setExample.add("Test")
  ```

- **Pop**

  Untuk menghapus dan return data pada set, data yang di pop pertama kali adalah integer pertama, setelah itu string pertama

  ```
  setExample.pop()
  ```

- **Remove**

  Untuk menghapus nilai yang sama yang terdapat pada set jika tidak terdapat nilai yang sama maka program akan error

  ```
  setExample.remove(1)
  ```

- **Discard**

  Untuk menghapus nilai yang sama yang terdapat pada set, jika tidak terdapat nilai yang sama maka program tidak melakukan apapun

  ```
  setExample.discard("Tes")
  ```

TUGAS PRAKTIKUM

1. Buatlah program yang mengalikan dua buah matriks 2x2

```
Input nilai matriks pertama index 1,1:1
Input nilai matriks pertama index 1,2:1
Input nilai matriks pertama index 2,1:2
Input nilai matriks pertama index 2,2:2
Input nilai matriks kedua index 1,1:1
Input nilai matriks kedua index 1,2:2
Input nilai matriks kedua index 2,1:2
Input nilai matriks kedua index 2,2:1
Hasil perkalian matriks
| 1, 1 | x | 1, 2 | = | 3, 3 |
| 2, 2 |   | 2, 1 |   | 6, 6 |
```

2. Buat program untuk mengubah detail data anda

```
Selamat datang untuk memulai silahkan input data anda
Input nama: Fatwa Anugah
Input umur: 21
Input alamat: Makassar
==================================================
Selamat datang Fatwa Anugah silahkan pilih opsi
==================================================
1. Detail anda
2. Ubah Nama
3. Ubah Umur
4. Ubah Alamat
5. Keluar
==================================================
Input opsi: 1
==================================================
Data anda adalah
Nama: Fatwa Anugah
Umur: 21
Alamat: Makassar
==================================================
Selamat datang Fatwa Anugah silahkan pilih opsi
==================================================
```

```
1. Detail anda
2. Ubah Nama
3. Ubah Umur
4. Ubah Alamat
5. Keluar
==================================================
Input opsi: 2
Silahkan Input nama baru: Fatwa Anugerah
Data anda sukses di diperbarui
==================================================
Selamat datang Fatwa Anugerah silahkan pilih opsi
==================================================
1. Detail anda
2. Ubah Nama
3. Ubah Umur
4. Ubah Alamat
5. Keluar
==================================================
Input opsi: 1
==================================================
Data anda adalah
Nama: Fatwa Anugerah
Umur: 21
Alamat: Makassar
==================================================
Input opsi: 5
==================================================
Selamat Tinggal Fatwa Anugerah
```

3. Buatlah program yang mencari duplikat dari 2 buah array

```
Input array ke 1: 1 2 3 4 5
Input array ke 2: 6 2 3 7 8
Terdapat 2 buah duplikat yaitu (2, 3)
```

Bab VI

Files

Files merupakan lokasi yang diberi nama pada disk untuk menyimpan informasi terkait. File digunakan untuk menyimpan data secara permanen pada *non-volatile* memory seperti hard disk. Saat kita ingin membaca atau menulis pada sebuah file, maka kita perlu membukanya terlebih dahulu. Setelah selesai, kita perlu menutupnya agar sumber daya yang terhubung ke file tersebut dibebaskan. Pada bahasa pemrograman Python, operasi pada file terjadi sesuai urutan berikut:

1. Membuka file
2. Membaca atau menulis
3. Menutup file

A. Penanganan File

Python memiliki *built-in function* yakni `open()` yang berfungsi untuk membuka sebuah file. Fungsi tersebut akan mengembalikan sebuah objek file. Fungsi tersebut menerima 2 argumen yaitu path dari file dan *mode* yang digunakan saat membuka file. Daftar mode yang tersedia dapat dilihat pada tabel berikut

Mode	Deskripsi
`"r"`	Membuka file untuk operasi baca (default)

`"w"`	Membuka file untuk operasi tulis. Membuat file baru apabila file tersebut tidak ada atau menghapus isi file bila file tersebut ada
`"x"`	Membuat file secara exclusive. Mengembalikan kesalahan jika file ada
`"a"`	Membuka file dan meletakkan kursor di akhir file tanpa menghapus isi file tersebut. Membuat file baru jika file tersebut tidak ada
`"t"`	Membuka file dalam *text mode* (default)
`"b"`	Membuka file dalam *binary mode*
`"+"`	Membuka file dengan operasi baca dan tulis

B. Membuka File

Untuk membuka file pada bahasa pemrograman Python, kita memanfaatkan fungsi **`open()`** yang secara default telah disediakan oleh Python. Penggunaan fungsi **`open()`** dapat dilihat pada syntax berikut

```
file = open("test.txt")
print(file.read())
file.close()
```

Output:

```
Hello world

This is the content of test.txt
```

Untuk membaca konten dari sebuah file, kita dapat menggunakan method read() dari objek file yang kita miliki. Perlu diperhatikan bahwa setelah kita selesai menggunakan objek file untuk membaca atau menulis sebuah file, kita perlu memanggil method close() dari objek file tersebut. Hal ini dilakukan untuk menutup file tersebut setelah selesai digunakan.

Terdapat cara lain untuk mengakses file pada Python tanpa perlu memanggil method close() secara eksplisit yakni menggunakan keyword with

```
with open("test.txt") as file:
    print(file.read())
```

Output:

```
Hello world

This is the content of test.txt
```

C. Menulis File

Untuk menulis file pada Python, kita menggunakan fungsi open() dengan menggunakan mode ‘w’.

```
with open(“buah.txt”, “w”) as file:
    file.write(“Apel\n”)
    file.write(“Mangga\n”)
    file.write(“Jeruk\n”)
    file.write(“Pisang\n”)
    file.write(“Jambu\n”)
```

Output :

```
Apel
Mangga
Jeruk
Pisang
Jambu
```

TUGAS PRAKTIKUM

1. Buatlah sebuah program untuk menyalin file dengan extensi .txt

   Format Input
   Baris pertama merupakan string yang menyatakan nama file (tanpa ekstensi file) yang akan disalin. Baris kedua, sebuah string yang menyatakan nama file hasil salinan.

   Format Output
   Tampilkan “Berhasil” apabila file berhasil disalin, atau “Gagal” apabila file gagal disalin.

   Contoh Input 1:

```
Latihan
Latihan-1
```

   Contoh Output 1:

```
Berhasil
```

   Latihan.txt

```
Baris pertama
Baris kedua
Dan Seterusnya...
```

   Latihan-1.txt

```
Baris pertama
Baris kedua
Dan Seterusnya...
```

2. Buatlah sebuah program untuk menyalin file dengan extensi .txt kemudian hasil salinannya ditulis kembali dengan format rata kanan.

   Format Input
   Baris pertama merupakan string yang menyatakan nama file (tanpa extensi file) yang akan disalin. Baris kedua, sebuah string yang menyatakan nama file hasil salinan tanpa ekstensi.

   Contoh Input 1:

```
Latihan
Latihan-1
```

Contoh Output 1:

```
Berhasil
```

Latihan.txt

```
Baris pertama
Baris kedua
Dan Seterusnya...
```

Latihan-1.txt

```
    Baris pertama
      Baris kedua
Dan Seterusnya...
```

Bab VII

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`.

A. MetaCharacters

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

1. [] – 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

2. . – 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.

3. ^ – 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)

4. **$ – 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

5. *** – 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

6. **+ – 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

7. **? – Question Mark**

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

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

8. **{m, n} – Braces**

RegEx ini akan mencari repetisi setidaknya **n** dan sebanyak **m** 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.

9. | – Alternation

Simbol | sama seperti operator or.

Ekspresi	String	Cocok ?
a|b	cde	Tidak cocok
	ade	1 cocok (pada **a**de)
	acdbea	3 cocok (pada **a**cd**b**e**a**)

10. () – Group

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

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

11. \ – 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.

B. 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 www.regex101.com

C. 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,

1. **`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)
```

   Output:

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

   Jika tidak terdapat kecocokan, maka akan dikembalikan list kosong.

2. **`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)
```

   Output:

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

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

3. **`re.sub()`**

   Sintaks dari **`re.sub()`** adalah:

```
re.sub(pattern, replace, string)
```

   Fungsi ini 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)
```

Output:

```
Hello1234Welcome4523Kaito1412
```

Jika pola tidak ditemukan, maka akan dikembalikan string asli.

4. **`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)
```

Output:

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

5. **`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`**.

```
import re

string = "Hello World"

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

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

TUGAS PRAKTIKUM

1. Buatlah sebuah program untuk mengecek sebuah string *S* apakah sesuai dengan kondisi berikut:
   - *S* memiliki panjang 45 karakter
   - 40 karakter pertama harus terdiri dari huruf uppercase atau lowercase atau digit bernilai genap
   - 5 karakter terakhir terdiri dari digit bernilai ganjil atau whitespace character.

   Format Input
   Sebuah string *S*

   Format Output
   Tampilkan "true" *S* memenuhi kondisi diatas, atau "false" *S* tidak memenuhi salah satu kondisi diatas.

   Contoh Input 1:

   ```
   2222222222aaaaaaaaaa2222222222aaaaaaaaaa13 57
   ```

   Contoh Output 1:

   ```
   true
   ```

   Contoh Input 2:

   ```
   x4202v2A22A8a6aaaaaa2G2222m222qwertyYuIo1395779
   ```

   Contoh Output 2:

   ```
   false
   ```

2. IP Address merupakan alamat unik yang mengidentifikasikan sebuah perangkat pada internet atau jaringan local. IP Address dapat dikelompokkan sebagai 2 yaitu IPv4 dan IPv6.

   IPv4 menggunakan alamat berukuran 32 bit yang dapat menghasilkan 2^{32} alamat berbeda. Format dari IPv4 adalah P.Q.R.S dimana masing masing P, Q, R dan S merupakan bilangan bulat dari 0 - 255.

   IPv6 berukuran 128 bit yang dapat dikelompokkan kedalam 8 grup dengan masing masing grup berukuran 16 bit. Setiap grup ditulis menggunakan 4 digit hexadecimal yang dipisahkan oleh tanda titik dua (:). 2001:0db8:0000:0000:0234:ff00:0036:752f

merupakan contoh dari IPv6. *Leading zeros* dapat saja diabaikan pada setiap grup. “....:5:.....” identik dengan “....:0005:.....”.

Buatlah program untuk mengecek apakah sebuah inputan merupakan IPv4, IPv6 atau bukan keduanya

Format Input

Sebuah integer N yang menyatakan jumlah baris inputan teks yang merupakan IPv4, IPv6 atau bukan keduanya. Jika teks tersebut merupakan IP Address maka tidak akan ada tambahan teks atau whitespace diawal atau diakhir IP Address tersebut. Setiap baris inputan tidak boleh melebihi 500 karakter

Format Output

N baris.
Output pada urutan ke-i harus bernilai “IPv4”, “IPv6”,”Bukan IP Address” tergantung dari nilai yang dideteksi

Contoh Input 1:

```
3
This line has trailing whitespace
121.203.197.20
2001:0db8:0000:0000:0000:ff00:0042:8329
```

Contoh Output 1:

```
Bukan IP Address
IPv4
IPv6
```

Contoh Input 2:

```
3
213.214.111.564
444.444.444.444 not an ip address
1050:0:0a:0:5:600:300c:326b
```

Contoh Output 2:

```
Bukan IP Address
Bukan IP Address
IPv6
```

Bab VIII

Object Oriented Programming

Sebagai bahasa pemrograman *multi-paradigm*, python mendukung pendekatan pemrograman yang berbeda. Salah satu pendekatan populer untuk memecahkan masalah pemrograman adalah dengan membuat objek yang dikenal sebagai *object oriented programming* atau pemrograman berorientasi objek.

Sebuah objek memiliki dua karakteristik:

- Atribut : variabel yang mewakili karakteristik objek
- Perilaku : method/fungsi pada suatu objek
- Class variable : variabel yang bersifat statik dan tidak diwariskan pada objeknya

Sebagai contoh, seekor burung beo adalah objek, karena memiliki sifat-sifat berikut:

- nama, umur, warna sebagai atribut
- bernyanyi, menari sebagai perilaku

OOP dalam Python bertujuan untuk mengimplementasikan entitas dalam dunia nyata seperti pewarisan, polimorfisme, enkapsulasi, dll.

Dalam Python, konsep OOP mengikuti beberapa prinsip dasar:

A. Class dan Objek

Class adalah blueprint untuk membuat objek. Kita dapat menganggap kelas sebagai sketsa burung beo dengan label. Ini berisi semua detail tentang nama, warna, ukuran, dll..

Contoh syntax untuk membuat kelas **burung**:

```
class Burung:
    pass
```

Kata kunci **`class`** untuk mendefinisikan kelas **burung** merupakan sebuah kelas kosong. Dari sebuah kelas, kemudian dapat dibuat instance. Instance adalah objek tertentu yang dibuat dari kelas tertentu. Instance akan memiliki setiap atribut dan perilaku dari class asalnya. Namun, atribut dari masing masing objek dari kelas yang sama bisa saja berbeda nilai atau karakteristiknya

Contoh syntax untuk membuat objek/instance dari class burung:

```
tobi = Burung()
```

```
tweety = Burung()
```

Operator assignment "=" juga digunakan untuk menyimpan objek dengan identifier di sisi kiri dan nama class di sisi kanan operator.

1. **Constructor**

   Konstruktor adalah fungsi yang dipanggil tepat setelah suatu objek dibuat. Fungsi ini dibuat di dalam blok utama kelas. Pada umumnya, constructor dibuat untuk melakukan inisialisasi nilai atribut dari suatu entitas objek.

   Contoh syntax untuk membuat konstruktor:

```
# class
class Burung:
     def __init__(self):
          print("objek berhasil dibuat!")
# objek
tobi = Burung()
```

   Output:

```
> "objek berhasil dibuat!"
```

   Selain **`__init__`** terdapat dua constructor lainnya yang memiliki fungsi tersendiri:

Constructor	Deskripsi
`__new__(cls)`	Dipanggil otomatis untuk membuat instance baru
`__init__(self)`	Dipanggil otomatis untuk inisialisasi instance baru
`__del__(self)`	Dipanggil otomatis ketika objek dihapus

2. **Menambahkan Atribut**

   Atribut adalah variabel Python yang hanya dimiliki oleh satu objek. Atribut dapat diakses dalam lingkup objek dan harus di inisialisasi dalam fungsi konstruktor kelas menggunakan keyword **`self.[nama atribut]`**. Atribut juga dapat ditambahkan langsung pada saat setelah objek dibuat dengan syntax **`[nama objek].[nama atribut]`**

   Contoh syntax untuk menambahkan atribut menggunakan konstruktor:

```
# class
class Burung:
     def __init__(self, jenis):
          Self.jenis = jenis
# objek
tobi = Burung("kolibri")
print(kolibri.jenis)
```

Output:

```
> kolibri
```

Contoh syntax untuk menambahkan atribut secara langsung:

```
# class
class Burung:
    pass

# objek
tobi = Burung()
tobi.jenis = "kolibri"
print(kolibri.jenis)
```

Output:

```
> kolibri
```

3. Menambahkan Perilaku/Method

Perilaku adalah tindakan yang dapat terjadi pada suatu objek. Perilaku yang dapat dilakukan pada kelas objek tertentu disebut *method*. Pada tingkat pemrograman, *method* seperti fungsi dalam pemrograman terstruktur, tetapi mereka secara ajaib memiliki akses ke semua data yang terkait dengan objeknya. Seperti fungsi, metode juga dapat menerima parameter dan mengembalikan nilai.

Contoh syntax untuk menambahkan atribut secara langsung:

```
# class
class Burung:
    def __init__(self, jenis):
        self.jenis = jenis
        self.kesehatan = 5
```

```
    def tidur(self, menit):
        self.kesehatan += menit/4
            print("tertidur selama " + str(menit) +
"menit")

# objek
kolibri = Burung("kolibri")
kolibri.tidur(30)
print("kesehatan :" + str(kolibri.kesehatan))
```

Output:

```
> tertidur selama 30 menit
> kesehatan : 12.5
```

B. Inheritance

Inheritance (Warisan) adalah cara membuat kelas baru dengan menggunakan detail kelas (atribut dan fungsi) yang ada tanpa memodifikasinya. Kelas yang baru terbentuk adalah kelas turunan (atau kelas **child**). Demikian pula kelas yang ada adalah kelas dasar (atau kelas **parent**).

```
# class parent
class Burung:
    def __init__(self, jenis):
        self.jenis = jenis
        self.kesehatan = 5

    def tidur(self, menit):
        self.kesehatan += menit/4
        print("tertidur selama " + str(menit) + " menit")

# class child
class Penguin(Burung): # inheritance ke class burung
    def __init__(self, jenis):
        super().__init__( jenis) # memanggil __init__ parent
```

```
    def berenang(self):
        print("penguin berenang dengan cepat!")

# objek
skipper = Penguin("penguin")
skipper.tidur(30)
print("kesehatan :" + str(kolibri.kesehatan))
print(skipper.jenis)
```

Output:

```
> tertidur selama 30 menit
> kesehatan : 12.5
> penguin
```

Berdasarkan contoh diatas, class **Penguin** dapat memanggil method dan atribut yang telah didefinisikan pada class **Burung** meskipun pada class penguin atribut dan method tersebut tidak didefinisikan. Jika method pada class child diwarisi dari class parent, namun berbeda halnya dengan constructor yang tidak dapat diwariskan. Namun, constructor class parent masih dapat dipanggil dengan menggunakan keyword **`super().__init__()`**.

C. Encapsulation

Menggunakan OOP di Python, kita dapat membatasi akses ke method dan variabel. Hal ini untuk mencegah data agar tidak dapat dimodifikasi langsung menggunakan yaitu dengan menggunakan konsep enkapsulasi. Dalam Python, untuk menunjukkan atribut private (pribadi) menggunakan garis bawah sebagai awalan yaitu _ tunggal atau ganda __.

```
class Komputer:

    def __init__(self):
        self.__hargaMax = 900

    def jual(self):
        print("menjual seharga : {}".format(self.__hargaMax))
```

```
    def setHargaMax(self, harga):
        self.__hargaMax = harga

komp = Komputer()
komp.jual()

# mengubah harga secara langsung
komp.__hargaMax = 1000
komp.jual()   # output = 900 (variabel private, tidak dapat
diubah
                            secara langsung)

# menggunakan fungsi set
komp.setHargaMax(1000)
komp.jual()  # output = 1000 (dapat diubah melalui method set)
```

Output:

```
> menjual seharga : 900
> menjual seharga : 900
> menjual seharga : 1000
```

D. Polymorphism

Polimorfisme adalah kemampuan (dalam OOP) untuk menggunakan interface publik untuk berbagai bentuk (tipe data). Misalkan, kita perlu mewarnai sebuah bentuk, ada beberapa pilihan bentuk (persegi panjang, persegi, lingkaran). Namun kita bisa menggunakan metode yang sama untuk mewarnai bentuk apapun. Konsep ini disebut Polimorfisme.

```
# class parent
class Burung:
    def __init__(self, jenis):
        self.jenis = jenis
        self.kesehatan = 5

    def terbang(self):
        print("Burung sedang terbang!")
```

```
# class child
class Penguin(Burung):
    def __init__(self, jenis):
        super().__init__(jenis)

    def terbang(self):
        print("Penguin tidak bisa terbang!")

# interface
def test_terbang(burung):
    burung.terbang()

# objek
tobi = Burung("kolibri")
skipper = Penguin("penguin")

# memanggil interface pada kedua objek
test_terbang(tobi)
test_terbang(skipper)
```

Output:

```
> Burung sedang terbang!
> Penguin tidak bisa terbang!
```

Dalam program di atas, didefinisikan dua kelas **tobi** dan **skipper**. Masing-masing memiliki method terbang() yang bersifat publik. Namun, fungsinya berbeda.

Untuk menggunakan konsep polimorfisme, dibuat suatu interface publik yaitu fungsi test_terbang() yang mengambil objek apa pun dan memanggil metode terbang() objek. Jadi, mengirim objek **tobi** dan **skipper** sebagai parameter dalam fungsi test_terbang(), fungsi tersebut tetap dapat bekerja.

TUGAS PRAKTIKUM

1. Buatlah sebuah class yang menerapkan konsep inheritance dan encapsulation, dan polymorphism.

2. Buatlah class lengkap yang memiliki atribut dan method berdasarkan diagram berikut. Kemudian buatlah dua objek dari class tersebut yang dapat berinteraksi menggunakan method yang ada.