Modul Praktikum
Algoritma dan Pemrograman
D4 Teknologi Rekayasa Multimedia

February 21, 2025

Daftar Isi

1 Instalasi Python **7**

2 Literal, Operator, Tipe Data, dan Variabel **9**
- 2.1 Literal . . . 10
- 2.2 Jenis/Tipe Data . . . 10
- 2.3 Mendefinisikan Variabel . . . 11
- 2.4 Tipe Data Dasar . . . 13
- 2.5 Tipe Data Lanjut . . . 14
  - 2.5.1 List . . . 15
  - 2.5.2 Tuple . . . 18
  - 2.5.3 Dictionary . . . 18
- 2.6 Operator . . . 20
- 2.7 Input . . . 21
- 2.8 Contoh Program . . . 22
- 2.9 Kerjakan . . . 22

3 Percabangan **27**
- 3.1 `if .. else` . . . 27
- 3.2 `if .. elif .. else` . . . 28
- 3.3 Blok . . . 29
- 3.4 `if .. in list` . . . 29
- 3.5 Contoh Program . . . 30
- 3.6 Kerjakan . . . 31

4 Pengulangan **33**
- 4.1 Pengulangan Menggunakan `for .. in range()` . . . 33
- 4.2 Counter . . . 33
- 4.3 Variasi Fungsi `range()` . . . 34
- 4.4 Pengulangan untuk List, Tuple, dan Dictionary . . . 35
- 4.5 Pengulangan untuk String . . . 36
- 4.6 `break` dan `continue` . . . 36

4.7 Pengulangan di Dalam Pengulangan . . . 37
4.8 Pengulangan Menggunakan `while` . . . 39
4.9 Kerjakan . . . 40

5 Latihan **43**
5.1 Variabel . . . 43
5.1.1 Medefinisikan Variabel . . . 43
5.1.2 Tipe Data Dasar . . . 43
5.1.3 Tipe Data Lanjut . . . 43
5.1.4 Operator . . . 44
5.1.5 Input . . . 45
5.2 Percabangan . . . 45
5.3 Pengulangan . . . 45
5.4 Membuat Program Sederhana . . . 46

6 Fungsi **49**
6.1 Fungsi . . . 49
6.2 File . . . 52
6.2.1 Cek File . . . 52
6.2.2 Membuat/Menghapus File . . . 52
6.2.3 Membuka/Menutup File . . . 53
6.2.4 Membaca File . . . 54
6.2.5 Menulis File . . . 56
6.3 Kerjakan . . . 57

7 Tipe Data Kustom **61**
7.1 Contoh Program . . . 63
7.2 Kerjakan . . . 66

8 Debugging dan Versioning **67**
8.1 Debugging . . . 67
8.2 Versioning . . . 70
8.2.1 Instalasi Git . . . 71
8.2.2 Inisialisasi Git . . . 72
8.2.3 Melacak Perubahan Kode (`add` dan `commit`) . . . 73
8.2.4 `Branch` . . . 75
8.2.5 GitHub . . . 76
8.3 Kerjakan . . . 78

9 Asesmen dan Proyek **79**
9.1 Asesmen 79
9.2 Proyek 1 81
9.3 Proyek 2 81
9.4 Contoh Program untuk Asesmen 82

Modul 1

Instalasi Python

Modul 2

Literal, Operator, Tipe Data, dan Variabel

1. **Tujuan**: Mengenal literal, operator, tipe data, dan variabel dalam Python
2. **Pengerjaan**: Perorangan
3. **Durasi**: 3 × 50 menit
4. **Perangkat**: Laptop dan Internet

Secara abstrak, program komputer dapat dilihat seperti Gambar 2.1. Program komputer memiliki input dan output. Ke dalam input, dimasukkan data. Dari output, keluar data. Jadi, tugas program komputer adalah memproses data. Dengan kata lain: dari data menjadi data.

Gambar 2.1: Secara abstrak, suatu program dianggap memiliki alur input-proses-output dengan objek yang melewati input dan output adalah data.

Oleh karena program berkaitan erat dengan data, kita mulai juga dengan data. Ada 3 istilah penting dalam pemrograman terkait data, yaitu

1. Literal
2. Jenis/tipe data
3. Operator

2.1 Literal

Literal adalah nilai/isi data. Contohnya bilangan `5`. Contoh lain adalah huruf atau sekumpulan karakter (string), misalnya `"a"` dan `"Bandung"`. Keduanya ditulis dalam tanda petik untuk membedakan dengan perintah yang dipakai dalam bahasa pemrograman. Di Kode 2.1, kata `print` yang pertama adalah perintah dan kata `"print"` di dalam tanda petik adalah literal.

Kode 2.1: Perintah dan literal

```
1 print("print")
```

Kita bisa menuliskan literal ke dalam Python Interpreter. Sebagai contoh, ketikkan literal-literal berikut dan tekan Enter setiap kali selesai mengetikkan satu literal.

1. `5`
2. `5.03`
3. `'a'`
4. `'algoritma'`
5. `True`
6. `[1,2,3]`

Sekarang, coba ketikkan `a` (tanpa tanda petik). Apa yang terjadi? Mengapa demikian?

2.2 Jenis/Tipe Data

Data tentu bermacam-macam. Ada data berupa bilangan bulat (Contoh `5`). Ada data berupa bilangan riil (Contoh `5.03`). Ada data berupa karakter(Contoh `"a"`). Ada data berupa rangkaian karakter (Contoh `"algoritma"`). Ada data berupa nila kebenaran suatu ekspresi/logika (Contoh `True`). Ada juga data berupa kumpulan bilangan (Contoh `[1,2,3]`).

Dalam pemrograman, setiap tipe data biasanya ada namanya. Sebagai contoh, untuk data berupa bilangan bulat, biasanya tipe datanya disebut **integer**. Untuk data berupa bilangan riil, disebut **float** atau **double**. Untuk karakter biasanya disebut **char**. Untuk serangkaian karakter biasanya disebut **string**. Untuk kumpulan bilangan bisa disebut sebagai **array** atau **list** atau **set** atau lainnya. Setiap bahasa pemrograman, menentukan sendiri tipe datanya. Pemrogram dapat juga membuat tipe data kustom sesuai kebutuhan.

Coba ketikkan setiap baris berikut ke dalam Python Interpreter (Enter setelah mengetikkan satu baris).

1. `type(5)`
2. `type(5.03)`
3. `type('a')`
4. `type('algoritma')`
5. `type(True)`
6. `type([1,2,3])`

Sebagai catatan, dalam bahasa pemrograman Python, tidak dibedakan antara tipe data **char** dan **string**. Keduanya dianggap sebagai **string**.

2.3 Mendefinisikan Variabel

Satu komponen penting dari sebuah program adalah variabel. Variabel digunakan untuk menampung data. Variabel memiliki nama agar mudah dipanggil. Contohnya adalah program di Kode 2.2.

Kode 2.2: Variabel dengan nama `a`

```
1 a = 100
2
3 print(a)
```

Di Kode 2.2, variabel dengan nama `a` diisi dengan data berupa bilangan 100. Kita bisa melambangkannya seperti ini: `a` ← 100. Baris ketiga dari Kode 2.2 adalah baris untuk mencetak isi dari variabel `a` ke layar (via terminal). Jika kamu menjalankan Kode 2.2, hasilnya adalah munculnya angka 100 di terminal. Berapakah bilangan yang muncul di kode-kode berikut?

Kode 2.3: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = 50
3 a = b
4
5 print(a) # bilangan berapa yang akan muncul?
```

Kode 2.4: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = 50
3 a = b
4 print (b) # bilangan berapa yang akan muncul?
```

Kode 2.5: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = a
```

```
3 print (b) # bilangan berapa yang akan muncul?
```

Kode 2.6: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = a
3 a = a + b
4 print (a) # bilangan berapa yang akan muncul?
```

Kode 2.7: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = a + b
3 print (b) # bilangan berapa yang akan muncul?
```

Kode 2.8: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = a
3 a = b + 100
4 print (a) # bilangan berapa yang akan muncul?
```

Kode 2.9: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = 50
3 print (a + b) # bilangan berapa yang akan muncul?
```

Kode 2.10: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = 50
3 c = 25
4 d = a + b + c
5
6 print (d) # bilangan berapa yang akan muncul?
```

Kode 2.11: Bilangan berapa yang akan muncul?

```
1 a = 100
2 b = a
3 c = a + b
4 a = 30
5 d = c + a + b
6 print (d) # bilangan berapa yang akan muncul?
```

2.4 Tipe Data Dasar

Jika kita perhatikan Kode 2.2-2.11, variabel-variabel di sana diisi bilangan bulat atau integer. Tentu saja variabel bisa diisi selain dari integer. Contohnya diisi *string* di Kode 2.12, diisi *float* di Kode 2.13, dan diisi *boolean* di Kode 2.14. Integer, string, float, dan boolean disebut sebagai tipe data. Lebih jauh, keempatnya merupakan tipe data dasar dalam Python.

Kode 2.12: Contoh variabel string

```
1 nama = "ismail"
2 print (nama)
```

Kode 2.13: Contoh variabel float

```
1 a = 1.4
2 print (a)
```

Kode 2.14: Contoh variabel boolean

```
1 hujan = True
2 print (hujan)
```

Berikut adalah beberapa contoh kode menggunakan tipe data string, float, dan boolean.

Kode 2.15: Contoh variabel string

```
1 nama_depan      = "Ismail"
2 nama_belakang   = "Rusli"
3 nama_lengkap    = nama_depan + " " + nama_belakang
4 print (nama_lengkap)
```

Kode 2.16: Contoh variabel float

```
1 a = 1.5
2 b = 3.4
3 c = a * b # a dikali b
4 d = a / b # a dibagi b
5 print (a, b)
```

Kode 2.17: Jika integer ditambah float, apa hasilnya?

```
1 a = 1       # integer
2 b = 2.3     # float
3 c = a + b   # apa tipe data c?
4 print (c)
```

Kode 2.18: Apa hasilnya?

```
a = 2
b = 3
c = a < b
print (c)
```

Kode 2.19: Apa hasilnya?

```
a = 4
b = 4
c = (a == b)
d = (a != b)
print (c, d)
```

Untuk mengetahui tipe data yang terdapat dalam sebuah variabel, kita dapat menggunakan perintah `type` seperti dalam Kode 2.20.

Kode 2.20: Mengetahui tipe data sebuah variabel

```
nama        = "budi"
umur        = 20
laki_laki   = True
ipk         = 3.9

print (type(nama))
print (type(umur))
print (type(laki_laki))
print (type(ipk))
```

Python adalah bahasa pemrograman yang cukup fleksibel. Saat membuat variabel, kita tidak perlu menentukan tipe data dari variabel tersebut. Bahkan, sebuah variabel dapat menampung tipe data yang berbeda-beda, seperti contoh di Kode 2.21.

Kode 2.21: Variabel `x` dapat berubah tipe datanya

```
x = "budi"  # tipe data string
print(x)

x = 100     # tipe data integer
print(x)
```

2.5 Tipe Data Lanjut

Selain tipe data dasar, seperti integer, float, string, dan boolean, terdapat juga tipe data lanjut. Berikut adalah beberapa contoh tipe data lanjut.

2.5.1 List

List dapat dianggap sebagai daftar. Sebuah list dapat berisi daftar integer, string, float, atau boolean. List juga dapat berisi daftar campuran.

Kode 2.22: Tipe data *list* dari integer

```
1 # variabel a berisi tipe data list
2 # yang berisi 6 integer
3 a = [1,2,3,4,5,6]
4 print (a)
```

Kode 2.23: Tipe data *list* dari float

```
1 a = [1.5, 2.3, 0.1, 7.9, -3.2]
2 print (a)
```

Kode 2.24: Tipe data *list* dari string

```
1 a = ['aku', 'adalah', 'indonesia']
2 print (a[0], a[1], a[2])
```

Kode 2.25: Tipe data *list* dari boolean

```
1 a = [True, False, False, True]
2 print (a)
```

Kode 2.26: Tipe data *list* dari data gabungan

```
1 a = [1, "nama", 3.7, True]
2 print (a)
```

Untuk mengambil satu data dari list, gunakan kurung siku seperti contoh Kode 2.27 berikut. Untuk mengakses data pertama dalam list, gunakan indeks 0. Untuk mengakses data terakhir dalam list, gunakan indeks -1 jika kita tidak tahu ada berapa elemen di dalam list tersebut.

Kode 2.27: Akses satu data dalam list

```
1 a = [1,2,3,4,5]
2 print (a[0], a[1])
```

Kode 2.28: Akses satu data dalam list

```
1 a = ['aku', 'adalah', 'indonesia']
2 print (a[0], a[1], a[2])
```

Kode 2.29: Akses data terakhir dalam list

```
1 a = [1,2,3,4,5]
2 print (a[-1])
```

Untuk menambahkan data ke akhir list, gunakan perintah `append`. Contohnya seperti dalam Kode 2.30. Sementara untuk menambah data di sembarang posisi, gunakan perintah `insert`. Contohnya di Kode 2.31.

Kode 2.30: Menambah data di akhir list

```
1 a = [1,2,3,4,5]
2 a.append (6)
3 print (a)
```

Kode 2.31: Menambah data di posisi sebelum data ke-1 di list

```
1 a = ["nama", "ismail"]
2 a.insert (1, "saya")
3 print (a)
```

Untuk menghapus data di list, gunakan perintah `remove`, `pop`, atau `del`. Berikut adalah contoh-contohnya.

Kode 2.32: Menghapus data/item dari list

```
1  a = [21,22,23,24,25]
2  a.remove(21) # hapus 21
3  print (a)
4
5  a = [21,22,23,24,25]
6  a.pop(2) # hapus indeks ke-2, yaitu 23
7  print (a)
8
9  a = [21,22,23,24,25]
10 del a[1] # hapus indeks ke-1, yaitu 22
11 print (a)
```

Dua list juga bisa digabungkan, contohnya di Kode 2.33.

Kode 2.33: Penggabungan 2 list

```
1 a = [1,2,3]
2 b = [4,5,6]
3 c = a + b
4 print (c)
```

Untuk mengetahui jumlah data dalam list, gunakan perintah `len` (Kode 2.34).

Kode 2.34: Mengetahui panjang list

```
1 a = [1,2,3,4,5,6]
2 panjang_a = len(a)
3
4 print (panjang_a)
```

Sebagai latihan, coba pahami kode-kode berikut.

Kode 2.35: Apa hasilnya?

```
a = [1,2,3,4,5]
b = [5,4,3,2,1]
c = (a == b)
print (c)
```

Kode 2.36: Apa hasilnya?

```
a = [1,2,3,4,5]
b = [a[1], a[3]]
print (b)
```

Kode 2.37: Apa hasilnya?

```
a = [1,2,3,4,5,6,7,8,9,10]
b = a[:3]
c = a[3:]
d = a[3:6]
print (b, c, d)
```

Kode 2.38: Apa hasilnya?

```
a = [1,2,3]
b = [4,5,6]
c = [a,b]
print (c)
print (c[0])
print (c[0][0])
```

Kode 2.39: Apa hasilnya?

```
a = [1,2,3,4,5,6,7,8,9,10]
a.pop (1)
b = a[3]
c = b + a[2]
a.insert (3, c)
print (a)
```

Kode 2.40: Apa hasilnya?

```
a = [1,2,3,4,5,6,7,8]
b = a + [1]
print (b)
```

2.5.2 Tuple

Tuple mirip seperti list hanya saja elemen di dalam tuple tidak dapat diubah. Tuple juga dituliskan dalam kurung biasa sementara list dituliskan dalam kurung siku. Sebagai contoh perhatikan Kode 2.41.

Kode 2.41: Data dalam tuple tidak dapat diubah.

```
1 a = (1,2,3,4,5)
2 print (a)        # print tuple
3 print (a[0])     # akses data tuple
4
5 a[0] = 100       # error, element tuple tidak dapat diubah
6 print (a[0])
```

Dua tuple dapat digabungkan dengan menggunakan simbol +. Perhatikan contoh di Kode 2.42.

Kode 2.42: Dua tuple digabungkan menggunakan operator +.

```
1 a = (1,2,3)
2 b = (4,5,6)
3
4 print (a + b)
```

Tuple digunakan untuk merepresentasikan data yang ukurannya sudah diketahui, misalnya posisi objek dalam ruang 3 dimensi $\rightarrow (x, y, z)$.

2.5.3 Dictionary

Dictionary, seperti artinya yaitu kamus. Dictionary adalah tipe data yang terdiri dari pasangan key dan value. Dalam kamus, misalnya kamus Indonesia-Inggirs, setiap entri terdiri dari 2 item, yaitu kata dan artinya. Dalam dictionary juga sama. Untuk jelasnya, perhatikan Kode 2.43.

Kode 2.43: Tipe data dictionary

```
1 daftar_nama_umur = {
2         'budi'  : 30,
3         'wati'  : 35,
4         'amran' : 35
5     }
6
7 print (daftar_nama_umur)
8 print (daftar_nama_umur['budi'])
```

Di Kode 2.43, kita lihat pasangan datanya bertipe string dan integer ('budi' : 30). Setiap pasang dalam satu dictionary, tidak harus memiliki tipe data yang sama dan key tidak harus string. Contohnya di Kode 2.44.

Kode 2.44: Tipe key dan value dalam dictionary tidak harus sama

```
data_anggota = {
        'nama'      : 'budi',                  # string : string
        'umur'      : 30,                      # string : integer
        'berat'     : 59.6,                    # string : float
        'gender'    : True,                    # string : bool
        'nilai'     : [9, 8.5, 8, 7.7, 10],    # string : list
        'posisi'    : (3.5, 2.0, 45.3)         # string : tuple
    }

print (data_anggota)
```

Untuk menambah, mengganti, atau menghapus data di dictionary, lihat Kode 2.45. Di Kode 2.45, ditunjukkan dua cara untuk menghapus data dari dictionary, yaitu menggunakan perintah `del` dan `pop`. Selain dari kedua perintah tersebut, terdapat perintah lain juga. Kamu dapat mengetahui perintah-perintah yang ada di Python dengan mencarinya di Google[1].

Kode 2.45: Menambah, mengganti, atau menghapus data di dictionary.

```
data_anggota = {
        'nama'      : 'budi',
        'umur'      : 30,
        'ttl'       : 'bandung, 20 januari 1900',
        'alamat'    : 'jl. dimanapun no. 100'
}

print ("===== print dictionary =====")
print (data_anggota, '\n')

data_anggota['nama'] = 'dudi'
print (" ===== ganti nama =====")
print (data_anggota, '\n')

data_anggota['gender'] = 'L'
print ("===== tambah gender =====")
print (data_anggota, '\n')

del data_anggota['alamat']
print ("===== hapus alamat menggunakan del =====")
print (data_anggota, '\n')

data_anggota.pop ('ttl')
print ("===== hapus ttl menggunakan pop =====")
print (data_anggota, '\n')

# '\n' digunakan untuk membuat baris baru
```

[1]Contohnya di sini: https://www.w3schools.com/python/default.asp

Untuk mendapatkan semua key dari dictionary, perhatikan Kode 2.46.

Kode 2.46: Mendapatkan semua key dalam sebuah dictionary.

```
1  data_anggota = {
2          'nama'       : 'budi',
3          'umur'       : 30,
4          'ttl'        : 'bandung, 20 januari 1900',
5          'alamat'     : 'jl. dimanapun no. 100'
6  }
7
8  key = data_anggota.keys()
9  print (type(key))
10 print (key)
```

2.6 Operator

Dua variabel dapat dioperasikan menggunakan operator (Kode 2.47).

Kode 2.47: Operator aritmatika

```
1  a = 5
2  b = 7
3  c = a + b
4  d = a - b
5  e = a / b
6  f = a * b
7  g = b % a # mod
8
9  print(a, b, c, d, e, f, g)
```

Kode 2.47 menunjukkan operator aritmatika. Jenis operator lainnya adalah operator perbandingan. Operator perbandingan digunakan untuk membandingkan 2 variabel. Perhatikan Kode 2.48.

Kode 2.48: Operator perbandingan

```
1  a = 5
2  b = 7
3  c = a < b # lebih kecil
4  d = a > b # lebih besar
5  e = a == b # sama dengan
6  f = a != b # tidak sama dengan
7  g = a <= b # lebih kecil atau sama dengan
8  h = a >= b # lebih besar atau sama dengan
9
10 print(a, b, c, d, e, f, g, h)
```

Untuk variabel tipe data string, simbol `+` dapat digunakan untuk menggabungkan dua string atau lebih (*concatenation*). Perhatikan Kode 2.49.

Kode 2.49: Penggabungan 2 variabel string menggunakan operator `+`

```
1 nama_depan = "Budi"
2 nama_belakang = "Kusnadi"
3 nama_lengkap = nama_depan + " " + nama_belakang
4 print(nama_lengkap)
```

Selain itu, ada juga operator logika. Contohnya di Kode 2.50.

Kode 2.50: Operator logika

```
1 a = True
2 b = False
3
4 c = a and b
5 d = a or b
6 e = not a
7
8 print(a, b, c, d, e)
```

2.7 Input

Salah satu yang membuat sebuah program interaktif adalah program tersebut dapat menerima input dari pengguna. Perhatikan contoh di Kode 2.51. Jalankan prorgram tersebut. Apa yang terjadi?

Kode 2.51: Program yang dapat menerima input dari pengguna

```
1 nama = input("Siapa nama Anda: ")
2 print("Salam kenal " + nama)
```

Sekarang perhatikan contoh program input lainnya di Kode 2.52. Jalankan program tersebut dan apa yang terjadi?

Kode 2.52: Program yang meminta pengguna input angka

```
1 a = input("Masukkkan angka pertama = ")
2 b = input("Masukkkan angka kedua = ")
3 c = a + b
4 print("Jumlah 2 angka yang anda masukkan adalah " + c)
```

Apakah tipe data dari variabel `a`, `b`, dan `c` dalam Kode 2.52? Ya, betul, tipe data ketiga variabel tersebut adalah string. Agar masukan pengguna dapat kita anggap sebagai bilangan, kita harus mengubahnya dulu (*type casting*). Perhatikan Kode 2.53, tulis dan jalankan program tersebut.

Kode 2.53: Variabel `a` dan `b`, kita ubah ke integer dan `c` kita ubah ke string

```
1 a = input("Masukkkan angka pertama = ")
2 b = input("Masukkkan angka kedua = ")
3 c = int(a) + int(b)
4 print("Jumlah 2 angka yang anda masukkan adalah " + str(c))
```

Perintah di baris 3 adalah menyuruh komputer untuk mengubah terlebih dahulu variabel `a` ke integer (`int(a)`), lalu variabel `b` ke integer (`int(b)`), kemudian menjumlahkan 2 integer tersebut dan menyimpannya di `c`.

2.8 Contoh Program

Kode 2.54 adalah contoh program sederhana memilih menu di restoran. Tulis program tersebut dan jalankan. Apa yang terjadi?

Kode 2.54: Program menu

```
1  print ("")
2  print ("====================================")
3  print("Selamat datang di restoran Sukelezat.")
4  print ("====================================")
5  print ("")
6
7  makanan = input("Silakan tuliskan
8          makanan yang kamu pesan: ")
9
10 minuman = input("Sekarang, silakan tuliskan
11         minuman yang kamu pesan: ")
12
13 print ("")
14 print("Terima kasih, pesanan Anda adalah "
15         + makanan + " dan " + minuman)
```

2.9 Kerjakan

1. Terdapat error di Kode 2.55 - Kode 2.62. Perbaikilah.

Kode 2.55: Perbaiki program berikut.

```
1 a = 100
2 b
3
4 print (b)
```

Kode 2.56: Perbaiki program berikut.

```
a = 100
b = 200

print (c)
```

Kode 2.57: Perbaiki program berikut.

```
a = 100
b = "seratus"

print (a + b)
```

Kode 2.58: Perbaiki program berikut.

```
a = 3.2
b = 2.0
c = a / (b - 2.0)

print (c)
```

Kode 2.59: Perbaiki program berikut.

```
a = [1,2,3]
a.append (7)

print (a[4])
```

Kode 2.60: Perbaiki program berikut.

```
a = (1,2,3,4,5)
a[0] = 8

print (a)
```

Kode 2.61: Perbaiki program berikut.

```
a = [1,2,3]
b = [4,5,6]
c = [7,8,9,a,b]

print (c[3][1])
print (c[2][1])
```

Kode 2.62: Perbaiki program berikut.

```
a = {"nama" : "budi", "umur" : 20}
print (a["budi"])
```

2. Buatlah program yang menerima input dari user berupa angka 0-9, lalu munculkan string yang bersesuaian. Misalnya, user memasukkan angka 1, maka print string "satu". **Petunjuk:** Gunakan list untuk menyimpan kata-katanya.

3. Komputer dapat menghasilkan bilangan acak seperti yang tertulis di Kode 2.63.

Kode 2.63: Komputer menghasilkan bilangan acak antara 1-100

```
1  import random
2
3  r = random.randint(1,100)
4  print(r)
5  r = random.randint(1,100)
6  print(r)
7  r = random.randint(1,100)
8  print(r)
9  r = random.randint(1,100)
10 print(r)
```

Buatlah suatu game sederhana. Program men-generate angka acak antara 0-100. User menebak angka yang di-generate komputer (inputkan angkanya). Munculkan skor tebakannya dengan cara menentukan selisih antara angka komputer dan angka tebakannya. Perhatikan contoh di Kode 2.64.

Kode 2.64: Contoh program tebak angka.

```
1  Tebak angka yang di-generate komputer: 50
2  Skor anda adalah :  34
3  Angka yang di-generate komputer adalah:  84
```

4. Suatu list dapat diubah menjadi tuple, carilah caranya di Internet lalu buat program contohnya. Buat juga sebaliknya, yaitu dari tuple menjadi list.

5. Buat program yang memunculkan nama 5 huruf secara random dengan ketentuan, huruf ke-2 dan ke-4 vokal, sisanya konsonan. Petunjuk: gunakan dua list, yaitu list untuk vokal dan list untuk konsonan. Lalu gunakan fungsi random untuk mengambil huruf dari kedua list tersebut.

6. Buat seperti program nama acak sebelumnya, tapi minta user untuk memasukkan huruf pertama nama acak tersebut.

7. Buat program pertandingan dengan skenario sebagai berikut.

(a) Ketika program dijalankan, user diminta memasukkan nama pertama, lalu enter
(b) Selanjutnya, user diminta memasukkan angka ke-2, lalu enter
(c) Lalu program menuliskan nama pertama dan skornya (random antara 0-100) serta nama kedua dan skornya (dalam baris berbeda)

8. Buatlah program yang mengacak kata. Skenarionya adalah, user diminta untuk memasukkan kata, lalu tampilkan versi teracak dari kalimat tersebut (kata-katanya diacak). Petunjuk: gunakan fungsi `split` untuk memisahkan kalimat ke dalam list dari kata-kata. Lalu gunakan `random.shuffle()` untuk mengacak list tersebut. Selanjutnya, gabungkan lagi kata-kata dalam list tersebut menggunakan perintah `" ".join(list)`

Modul 3

Percabangan

1. **Tujuan**: Mengenal percabangan dalam Python
2. **Pengerjaan**: Perorangan
3. **Durasi**: 3 × 50 menit
4. **Perangkat**: Laptop dan Internet

3.1 `if .. else`

Salah satu fitur dalam Python adalah percabangan. Percabangan digunakan untuk menentukan baris yang dikerjakan selanjutnya. Perhatikan Kode 3.1 lalu lengkapi Kode 3.2.

Kode 3.1: Percabangan dalam Python

```
1 nilai = 81
2
3 if nilai >= 80:
4     print ("A")
5 else:
6     print ("tidak A")
```

Di Kode 3.1 di baris 3, terdapat suatu percabangan. Kode akan terpecah tergantung dari suatu kondisi, yaitu

1. Jika variabel `nilai` isinya lebih atau sama dengan 80, program dilanjutkan ke baris 4.
2. Jika kebalikannya (artinya kurang dari 80), program dilanjutkan ke baris 6.

Kode 3.2: Percabangan dalam Python

```
1 nilai = 100
```

```
2
3 # jika nilai lebih dari 100
4 # tulis "nilai lebih dari 100"
5 # jika nilai tidak lebih dari 100
6 # tulis "nilai tidak lebih dari 100"
```

3.2 if .. elif .. else

Jika percabangannya banyak, bisa digunakan perintah `if ... elif ... else` seperti di Kode 3.3.

Kode 3.3: Percabangan dalam Python

```
1 nilai = 75
2
3 if nilai >= 80: print ("A")
4 elif nilai >= 70: print ("AB")
5 elif nilai >= 65: print ("B")
6 elif nilai >= 60: print ("BC")
7 elif nilai >= 50: print ("C")
8 elif nilai >= 40: print ("D")
9 else: print ("E")
```

Kode 3.3 dapat digambarkan sebagai diagram alir (flowchart) seperti ditunjukkan di Gambar 3.1.

Coba lengkapi Kode 3.4 berikut.

Kode 3.4: Percabangan dalam Python

```
1 umur = 45
2
3 # jika umur lebih dari 40 disebut dewasa,
4 # jika antara 20 - 40 disebut muda,
5 # jika antara 10 - 20 disebut remaja
6 # di bawah 10 disebut anak-anak
```

Perhatikan Kode 3.5 dan Kode 3.6. Coba tulis dan bandingkan hasilnya. Adakah perbedaan kedua kode tersebut?

Kode 3.5: Percabangan dalam Python

```
1 kuliah = 'alpro'
2
3 if kuliah == 'alpro':
4     print (kuliah)
5 elif kuliah == 'pbo':
6     print ('bukan alpro tapi juga ngoding')
7 elif kuliah == 'agama':
8     print ('sama sekali gak ngoding')
```

Kode 3.6: Percabangan dalam Python

```
1 kuliah = 'alpro'
2
3 if kuliah == 'alpro':
4     print (kuliah)
5 if kuliah == 'pbo':
6     print ('bukan alpro tapi juga ngoding')
7 if kuliah == 'agama':
8     print ('sama sekali gak ngoding')
```

3.3 Blok

Setiap pilihan kondisi dalam percabangan, mengarahkan program untuk mengeksekusi suatu blok (sebaris program). Dalam Python, satu blok program dicirikan dengan indentasi. Sebagai contoh, perhatikan Kode 3.7.

Kode 3.7: Blok ditandai dengan indentasi

```
1  nama = "ismail"
2  umur = 45
3  gender = "L"
4
5  if nama == "ismail":
6      print (nama)
7      print (umur)
8      print (gender)
9  else:
10     print ("nama tidak dikenal")
11     print ("umur tidak dikenal")
12     print ("gender tidak diketahui")
```

Di Kode 3.7, jika variabel `nama` isinya "ismail", maka setelah baris 5, program akan mengeksekusi baris 6–8. Sementara jika `nama` isinya bukan "ismail", setelah baris 5, program akan loncat ke baris 10–12.

3.4 `if .. in list`

Kata kunci `if` dapat digunakan untuk mencari item di dalam list. Perhatikan Kode 3.8.

Kode 3.8: `if` untuk mencari item di dalam `list`

```
1 buahbuahan = ['mangga','jeruk','duren','pepaya','jambu']
2
3 if 'mangga' in buahbuahan:
```

```
4       print ('ada mangga dalam list')
5   else:
6       print ('tidak ada mangga dalam list')
7
8   if 'semangka' in buahbuahan:
9       print ('ada semangka dalam list')
10  else:
11      print ('tidak ada semangka dalam list')
```

3.5 Contoh Program

Untuk lebih memahami percabangan, tuliskan dan pahami kode-kode berikut.

Kode 3.9: Percabangan dalam Python

```
1   a = 100
2   b = 200
3
4   if a > b:
5       print ("A lebih besar dari B")
6   elif a < b:
7       print ("A kurang dari B")
8   else:
9       print ("A sama dengan B")
```

Kode 3.10: Percabangan dalam Python

```
1   a = 100
2   b = 200
3
4   c = (a == b) # apa isi C? True atau False?
5
6   if c:
7       print ("A sama dengan B")
8   else:
9       print ("A tidak sama dengan B")
```

Kode 3.11: Percabangan dalam Python

```
1   import random
2
3   angka = random.randint (1,5)
4   tebakan = input ("Tebak angka yang di-generate komputer: ")
5   tebakan = int(tebakan)
6
7   if (tebakan == angka):
```

```
8       print ("Anda hebat, tebakan anda benar!!")
9   else:
10      print ("Tebakan anda tidak tepat")
11      print ("Angka yang benar adalah " + str (angka))
```

Ada beberapa hal di Kode 3.11 yang perlu diperhatikan.

1. Baris 5: di baris ini, variabel `tebakan` yang tadinya berisi data string, diubah menjadi data integer.
2. Baris 7: ekspresi setelah `if` boleh diberi kurung.
3. Untuk setiap blok kode (misal blok `if` dan juga blok `else`), baris-baris dalam kode dituliskan menjorok. Jadi, jika blok `if` yang dijalankan, maka jalankan baris 8. Sementara jika blok `else` yang dijalankan, maka jalankan baris 10 dan 11.

3.6 Kerjakan

1. Buatlah program berikut. User input 10 nama, lalu kelompokkan berdasarkan huruf awal. Masukkan setiap kelompok ke dalam list.

2. Tebak angka. Jika selisih di bawah 10, tulis "menang". Jika di antara 10–30, tulis "hampir". Jika di atas 30, tulis "jauh".

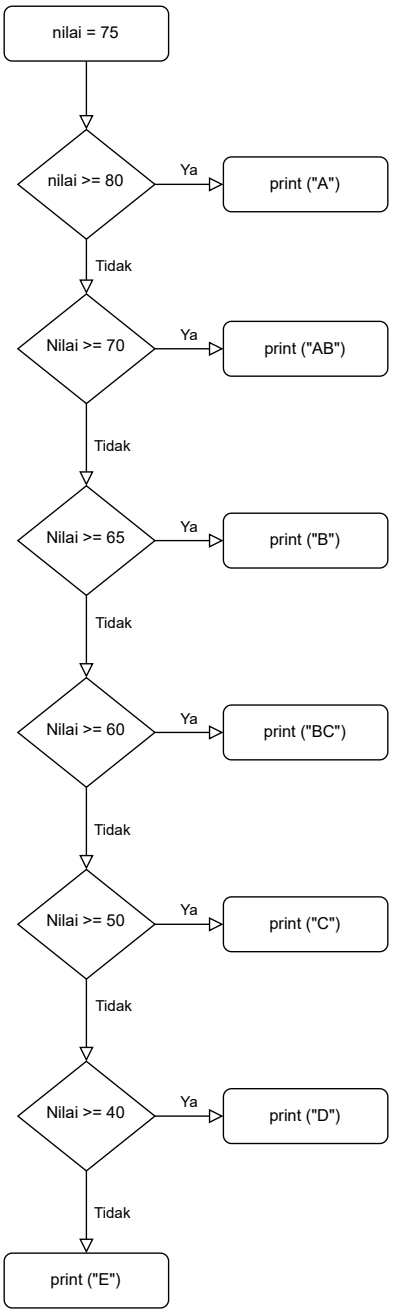
3. Tampilkan 10 barang, lalu minta user memilih barang dengan memasukkan angka. Setelah itu, tunjukkan harga barang. Jika memasukkan angka yang tidak sesuai, beri peringatan.

4. Buat kalkulator. User memasukkan dua angka, lalu memilih operasi. Keluarkan hasilnya.

5. Minta user memasukkan 3 angka. Munculkan ke-3 angka secara terurut.

6. Buat sebuah list nama. Minta user memasukkan nama. Lalu cari apakah nama yang dimasukkan ada dalam list atau tidak.

7. Buat tampilan menu makanan dan minuman dengan harga. Minta user memilih 1 makanan dan 1 minuman. Totalkan harganya.



Gambar 3.1: Diagram alir (flowchart) untuk menjelaskan Kode 3.3.

Modul 4

Pengulangan

1. **Tujuan**: Mengenal pengulangan dalam Python
2. **Pengerjaan**: Perorangan
3. **Durasi**: 4 × 50 menit
4. **Perangkat**: Laptop dan Internet

4.1 Pengulangan Menggunakan `for .. in range()`

Satu fitur penting dalam pemrograman adalah pengulangan. Pengulangan digunakan untuk mengulangi baris-baris kode secara berulang-ulang. Perhatikan Kode 4.1.

Kode 4.1: Pengulangan 10 kali

```
1 for i in range(10):
2     print ("ini diprint 10x")
```

Di Kode 4.1, baris 1 adalah perintah untuk melakukan pengulangan sebanyak 10 kali. Apa yang diulang? Yang diulang adalah baris 2 (baris yang menjorok ke dalam). Coba kerjakan Kode 4.2.

Kode 4.2: Kerjakan

```
1 # Minta user memasukkan angka
2 # Lalu print "Kalimat ini diprint berulang"
3 #   sebanyak angka yang dimasukkan user
```

4.2 Counter

Di Kode 4.1, variabel `i` adalah variabel counter yang nilainya akan berubah setiap kali pengulangan. Untuk jelasnya, perhatikan Kode 4.3.

Kode 4.3: Variabel `i` berfungsi sebagai counter

```
1 pengulangan = 10
2
3 for i in range(pengulangan):
4     print (i)
```

4.3 Variasi Fungsi `range()`

Fungsi `range()` dapat diikuti dengan 1 angka, 2 angka, atau 3 angka (data yang mengikuti fungsi disebut argumen atau parameter). Jika diisi 1 parameter (`range(x)`) maka counter akan bergerak dari 0 sampai `x`. Jika diisi 2 parameter (`range(x,y)`) maka counter akan bergerak dari `x` sampai `y`. Jika diisi 3 parameter (`range(x,y,z)`) make counter akan bergerak dari `x` sampai `y` dengan counter loncat sebesar `z`. Perhatikan Kode 4.4 dan Kode 4.5 untuk lebih jelasnya.

Kode 4.4: Variasi dalam `for .. in range()`

```
1 for i in range(10,0,-1): #(awal, akhir, step)
2     print (i)
```

Kode 4.5: Variasi dalam `for .. in range()`

```
1 for i in range(1,20,2):
2     print (i)
```

Kode 4.6: Kerjakan

```
1 for i in range(10):
2     # Print angka 1 sampai 10
```

Kode 4.7: Kerjakan

```
1 # Buatlah program yang menerima 3 angka dari user
2 # Gunakan angka pertama sebagai awal
3 # Gunakan angka kedua sebagai akhir
4 # Gunakan angka ketiga sebagai langkah (step)
5 # Misal: 1,20,3
6 # Maka program akan mencetak angka 1, 4, 7, 10, 13, 16, 19
7 # yang diprint ke bawah
```

4.4 Pengulangan untuk List, Tuple, dan Dictionary

Pengulangan juga dapat digunakan untuk memanggil setiap item yang ada di dalam list, tuple, atau dictionary. Sebagai contoh, perhatikan Kode 4.8-4.10.

Kode 4.8: Pengulangan dalam list

```
1 daftar = [1,2,3,4,5]
2
3 for angka in daftar:
4     print (angka)
```

Kode 4.9: Pengulangan dalam tuple

```
1 posisi = (1,2,3)
2
3 for x in posisi:
4     print (x)
```

Kode 4.10: Pengulangan dalam dictionary

```
1 daftar = {'nama': 'ismail',
2           'umur': 45,
3           'alamat': 'bandung'}
4
5 for key, value in daftar.items():
6     print (key, value)
```

Sebagai latihan, kerjakan Kode 4.11-4.13.

Kode 4.11: Kerjakan

```
1 daftar1 = [1,2,3,4,5]
2
3 for angka in daftar:
4     # buatlah list baru yang isinya adalah
5     # setiap angka di daftar1 + 5
6     # --> [6,7,8,9,10]
```

Kode 4.12: Kerjakan

```
1 daftar1 = [1,2,3,4,5]
2
3 for angka in daftar:
4     # buatlah list baru yang isinya
5     # hanyalah 3 item pertama
6     # dari daftar1, yaitu [1,2,3]
7     # gunakan 'if'
```

Kode 4.13: Kerjakan

```
daftar1 = [1,2,3,4,5]

for angka in daftar:
    # buatlah list baru yang isinya
    # kata ganjil atau genap
    # sesuai dengan angka di daftar1:
    # ['ganjil','genap','ganjil','genap','ganjil']

    # gunakan operator modulus
    # untuk cek ganjil genap
    # angka % 2 == 0 maka genap
    # angka % 2 != 0 maka ganjil
```

4.5 Pengulangan untuk String

Pengulangan juga dapat dilakukan untuk string. Perhatikan Kode 4.14-4.17.

Kode 4.14: Pengulangan

```
matakuliah = "algoritma dan pemrograman"

for huruf in matakuliah:
    print (huruf)
```

Kode 4.15: Pengulangan

```
matakuliah = "algoritma dan pemrograman"

for huruf in matakuliah:
    print (huruf, end = '')
```

4.6 `break` dan `continue`

Salah satu perintah penting dalam pengulangan adalah `break` dan `continue`. `break` digunakan untuk keluar dari loop. Sementara `continue` digunakan untuk lanjut ke iterasi selanjutnya dalam loop dengan mengabaikan kode di sisa loop. Perhatikan Kode 4.16-4.18.

Kode 4.16: Pengulangan

```
matakuliah = "algoritma dan pemrograman"

for karakter in matakuliah:
    if karakter == ' ':
```

```
5           break
6       print (karakter, end = '')
7
8   print ('')
```

Kode 4.17: Pengulangan

```
1   matakuliah = "algoritma dan pemrograman"
2
3   for karakter in matakuliah:
4       if karakter == ' ':
5           continue
6       print (karakter, end = '')
7
8   print ('')
```

Kode 4.18: Kerjakan

```
1   nama = input ("Silakan masukkan nama panjang: ")
2
3   vokal = []
4   konsonan = []
5
6   # lakukan loop
7   # kumpulkan vokal ke list vokal
8   # kumpulkan konsonan ke list konsonan
9   # lalu hitung ada berapa vokal
10  # dan ada berapa konsonan
```

4.7 Pengulangan di Dalam Pengulangan

Jika diperlukan, kita bisa membuat suatu pengulangan di dalam pengulangan. Perhatikan Kode 4.19-4.21.

Kode 4.19: Pengulangan

```
1   angka = [1,2,3,4,5]
2   huruf = ['a','b','c','d','e']
3
4   for x in angka:
5       for y in huruf:
6           karakter = str(x) + y
7           print (karakter, end = ' ')
8
9   print ('')
```

Kode 4.20: Pengulangan

```
bintang = '*'

for i in range(1,11):
    for j in range (i):
        print (bintang, end = '')
    print ('')
```

Kode 4.21: Pengulangan

```
bintang = '*'
baris_max = 11

for i in range (1,11):
    for j in range (baris_max - i):
        print (bintang, end = '')
    print ('')
```

Kode 4.22: Pengulangan

```
bintang = '*'
baris = 10

for i in range(1, baris):
    if i < 6:
        for j in range (i):
            print (bintang, end = '')
    else:
        for j in range (baris - i):
            print (bintang, end ='')

    print ('')
```

Kode 4.23: Pengulangan

```
bintang = '*'
baris = 10

for i in range(1, baris):
    if i < 6:
        for j in range (6 - i):
            print (bintang, end = '')
    else:
        for j in range (i - 4):
            print (bintang, end ='')

    print ('')
```

Sebagai latihan, kerjakan Kode 4.24

Kode 4.24: Kerjakan

```
# tambahkan kode berikut sehingga di setiap baris
# terdapat nomor barisnya
# 1 *
# 2 **
# 3 ***
# dst.

bintang = '*'

for i in range(1,11):
    for j in range (i):
        print (bintang, end = '')
    print ('')
```

Kode 4.25: Kerjakan

```
bintang = '*'
plus = '+'

for i in range(1,8):
    if i < 5:
        for j in range (5-i):
            print (bintang, end = '')
        for j in range (i):
            print (plus, end = '')
    else:
        # lengkapi bagian else ini
        # sehingga outputnya seperti berikut
        # *****+
        # *****+
        # *****+
        # *****+
        # *****+
        # *****+
        # *****+

    print ('')
```

4.8 Pengulangan Menggunakan `while`

Selain menggunakan perintah `for`, pengulangan dalam Python juga dapat menggunakan perintah `while`. Perhatikan Kode 4.26.

Kode 4.26: Pengulangan menggunakan perintah `while`

```
iterasi = 1
```

```
2 while iterasi < 5:
3     print("***")
4     iterasi = iterasi + 1
```

4.9 Kerjakan

1. Buat program yang menampilkan menu seperti berikut di Kode 4.27

Kode 4.27: Menu

```
1 ##############################
2 1. Pilihan 1
3 2. Pilihan 2
4 3. Pilihan 3
5 4. Keluar
6 ##############################
7 Masukkan pilihan Anda:
```

Jika user menekan 1, keluar tulisan "Anda memilih angka 1. Tekan Enter untuk melanjutkan" Setelah menekan Enter, layar kembali ke menu utama. Jika user menekan 4 (keluar), tuliskan "Terima kasih". Gunakan Kode 4.28 sebagai kerangka program.

Kode 4.28: Kerangka program

```
1  import os
2
3  # perintah untuk membersihkan layar
4  os.system ('clear')
5
6  # untuk Windows, ganti 'clear' dengan 'cls'
7
8  print ('########################')
9  print ('1. Pilihan 1')
10 print ('2. Pilihan 2')
11 print ('3. Pilihan 3')
12 print ('4. Keluar')
13 print ('########################')
14
15 pilihan = input ("Masukkan pilihan Anda: ")
16
17 if pilihan == '1':
18     pass
19     # print sesuatu
20     input() # program berhenti sebentar menunggu
21             # user menekan enter
22 elif pilihan == '2':
```

```
23      pass
24      # print sesuatu
25  elif pilihan == '3':
26      pass
27      # print sesuatu
28  elif pilihan == '4':
29      print ("Keluar")
30  else:
31      print ("Anda tidak memilih dengan benar")
```

2. Buat program yang meminta input 10 nama dari user, lalu kelompokkan berdasarkan huruf awalnya. Gunakan kerangka Kode 4.29.

Kode 4.29: Kerangka program

```
1  daftar_nama = {}
2
3  for i in range (10):
4      nama = input ("Masukkan nama ke-"+str(i+1)+": ")
5      huruf_awal = nama[0]
6
7      if huruf_awal in daftar_nama:
8          # jika sudah ada nama berawalan huruf_awal
9          # gunakan append
10     else:
11         # jika belum ada nama berawalan huruf_awal
12         daftar_nama[huruf_awal] = [nama]
13
14 # print hasilnya dengan rapi (gunakan pengulangan)
```

3. Buat program yang outputnya seperti dalam Kode 4.30.

Kode 4.30: Output

```
1  *****
2  ****
3  ***
4  **
5  *
6  $$
7  $$$
8  $$$$
9  $$$$$
```

4. Buat program menebak angka acak antara 1-100. User boleh menebak secara berulang-ulang. Setiap kali menebak, program memberi respon "terlalu besar" atau "terlalu kecil". Jika benar, program mengeluarkan pesan "Tepat!".

Modul 5

Latihan

Kerjakan masing-masing perintah berikut pertama-tama tanpa melihat referensi apapun. Jika mentok, cari tahu lewat Google.

5.1 Variabel

5.1.1 Medefinisikan Variabel

1. Buatlah 1 variabel dan isi dengan data.
2. Buatlah 2 variabel dan isi dengan data.
3. Buatlah 2 variabel dan isi masing-masing dengan data yang berbeda. Lalu, pindahkan isi satu variabel ke variabel yang lain.

5.1.2 Tipe Data Dasar

4. Buatlah 4 variabel dan isi masing-masing dengan data bertipe integer, string, float, dan boolean. Cetak ke layar tipe masing-masing variabel.
5. Buatlah 2 variabel berisi integer. Buat variabel ke-3 yang isinya penjumlahan dari 2 variabel pertama.
6. Buatlah 1 variabel, isi dengan integer. Ubahlah data integer tersebut menjadi string. Lakukan sebaliknya, yaitu string menjadi integer.
7. Buatlah dua variabel yang masing-masing berisi string. Gabungkan kedua string tersebut, lalu cetak ke terminal.

5.1.3 Tipe Data Lanjut

Mulai dari sini dan selanjutnya, jika disebutkan untuk membuat suatu data (misalnya "Buatlah `list`", maksudnya adalah buatlah variabel dan isi dengan `list`).

`List`

8. Buat contoh `list` dari integer.
9. Buat contoh `list` dari string.
10. Buat contoh `list` dari float.
11. Buat contoh `list` dari boolean.
12. Buat `list` berisi 5 integer. Lalu cetak ke layar.
13. Buat `list` berisi 5 integer. Lalu cetak data ke-2 dan ke-3 dari `list` tersebut.
14. Buat `list` berisi 5 integer. Lalu, tambahkan 1 lagi integer.
15. Buat `list` berisi 5 integer. Lalu, hapus 1.
16. Buat `list`, lalu cetak jumlah item dalam `list` tersebut.
17. Buat `list`, lalu cetak data terakhir ke layar.
18. Buat 2 `list`, lalu gabungkan. Cetak ke layar hasil penggabungannya.

`Tuple`

19. Buat contoh `tuple`.
20. Buat `tuple` berisi 3 integer, lalu cetak data ke-2.

`Dictionary`

21. Buat contoh `dictionary`.
22. Buat `dictionary` dengan 2 item. Lalu cetak key dan value dari item.
23. Buat `dictionary` dengan 2 item. Lalu cetak jumlah item dalam dictionary ke layar.
24. Buat `dictionary` lalu tambahkan item.
25. Buat `dictionary` lalu hapus item.
26. Buat `dictionary` lalu edit salah satu key item.
27. Buat `dictionary` lalu edit salah satu value item.

5.1.4 Operator

28. Buat contoh penggunaan operator aritmatika (penjumlahan, pengurangan, perkalian, dan pembagian).
29. Buat contoh penggunaan operator perbandingan (lebih besar, lebih besar sama dengan, lebih kecil, lebih kecil sama dengan, sama dengan, tidak sama dengan).
30. Buat contoh penggunaan operator logika (and, or, dan not).

5.1.5 Input

31. Buat perintah input sehingga jika dijalankan akan meminta user memasukkan angka.
32. Buat perintah input sehingga jika dijalankan akan meminta user memasukkan string.
33. Buat 2 perintah input untuk menerima 2 angka dari user. Jumlahkan kedua angka dan cetak hasil penjumlahannya ke layar.

5.2 Percabangan

34. Buat contoh kode dengan 2 percabangan. Misalnya, cabang jika benar dan cabang jika salah.
35. Buat contoh kode dengan 3 percabangan. Misalnya cabang jika `angka` > 100, jika 50 < `angka` <= 100, dan jika 0 <= `angka` <= 50.
36. Buat contoh kode dengan 4 percabangan. Misalnya jika huruf pertama string adalah 'a', 'b', 'c', atau lainnya.
37. Buat contoh kode dengan 2 percabangan. Buat masing-masing cabang mengeksekusi 3 baris kode.
38. Buat contoh kode untuk menguji jika suatu item berada di dalam sebuah `list`.
39. Buat contoh kode untuk menguji jika sebuah key berada di dalam sebuah `dictionary`.
40. Buat contoh kode untuk menguji jika value dari sebuah key adalah benar. Misalnya terdapat item `{'umur' : 40}` di dalam sebuah dictionary. Buat kode untuk cek jika umur adalah 40.

5.3 Pengulangan

41. Cetak kalimat "Saya senang pemrograman" sebanyak 100 kali ke layar menggunakan perintah `for`.
42. Cetak angka 0 sampai 99 ke layar (ke bawah).
43. Cetak angka 1 sampai 100 ke layar (ke samping).
44. Cetak angka 2–100 ke layar loncat 2 (ke samping).
45. Cetak angka 100–0 ke layar mundur (ke samping)
46. Buat `list` 5 item, lalu cetak masing-masing item ke layar (ke bawah).
47. Minta user memasukkan nama, lalu print masing-masing huruf ke bawah.
48. Buat dua list masing-masing berisi 5 item angka dan 5 item huruf. Lalu cetak semua kombinasi gabungan angka dan huruf ke layar (ke bawah).

49. Buat program yang mencetak 10 baris ke bawah. Setiap baris mencetak karakter bintang sebanyak angka barisnya. Misalnya, baris ke-1 mencetak 1 bintang, baris ke-2 mencetak 2 bintang (ke samping), dan seterusnya.
50. Buat pengulangan tak hingga menggunakan perintah `while`. Untuk menghentikannya, ketik ctrl-c di terminal.

5.4 Membuat Program Sederhana

51. Buat program yang meminta user memasukkan nama. Lalu program mencetak kalimat `"Hallo [nama], senang berkenalan dengan Anda"`. ganti `[nama]` dengan nama yang diinput user.
52. Buat program yang meminta user memasukkan 5 nama, lalu setelah selesai, cetak ke-5 nama tersebut. Contoh keluaran: `"Anda telah memasukkan 5 nama berikut, yaitu [nama1], ..., [nama5]"`.
53. Buat program yang telah menyimpan daftar 10 nama. Lalu minta user memasukkan satu nama. Jika nama ada dalam daftar, cetak `"Nama tercantum dalam daftar."`. Jika tidak, cetak `"Nama tidak tercantum dalam daftar."`.
54. Buat program yang meminta user memasukkan biodata, yaitu nama, umur, gender, tempat lahir, tahun lahir, dan alamat. Simpan ke dalam dictionary, lalu cetak datanya ke layar.
55. Buat program yang menampilkan menu 1-5. User dapat memilih dengan memasukkan angka 1-5. Jika user memilih 1-4, print menu yang dipilih. Setelah itu, program kembali ke menu utama. Jika memilih 5, keluar.
56. Buat program untuk menghitung sisi miring segitiga siku-siku. Minta user memasukkan panjang kedua sisi segitiga, lalu print panjang sisi miringnya. Gunakan fungsi `math.sqrt()` untuk menghitung akar. (import terlebih dahulu modul `math`)
57. Buat program yang menyebutkan suatu angka ganjil atau genap. Angka dimasukkan oleh user. Jika user tidak memasukkan angka, print "Anda tidak memasukkan angka.".
58. Buat program simulasi menebak letak koin, di tangan kiri atau kanan.
59. Buat program generate nama secara acak. Minta user untuk memasukkan jumlah huruf untuk nama yang akan digenerate secara acak. Selain itu, minta user untuk memilih jika nama akan dimulai dengan vokal atau konsonan. Minta juga user menentukan huruf pertamanya. Setelah itu, generate nama secara acak, dengan pola konsonan dan vokal yang saling bergantian.

60. Buat program menebak angka antara 1-100. User dapat menebak berulang-ulang. Jika tebakannya terlalu kecil, print "terlalu kecil". Jika tebakannya terlalu besar, print "terlalu besar". Jika tepat, print "tepat". Setiap kali menebak, user harus membayar koin 10. Berikan koin awal sebanyak 100. Jika tepat menebak, user mendapatkan 50 koin. Setelah tepat menebak, user diberi pilihan, menebak lagi atau keluar.

Modul 6

Fungsi

1. **Tujuan**: Mengenal fungsi dalam Python
2. **Pengerjaan**: Perorangan
3. **Durasi**: 4 × 50 menit
4. **Perangkat**: Laptop dan Internet

6.1 Fungsi

Perhatikan Kode 6.1 berikut. Kode 6.1 adalah program yang mencetak 10 baris keluaran. Setiap baris mencetak karakter bintang sebanyak posisi barisnya. Jadi, misalnya baris ke-1, akan mencetak karakter `*`. Baris ke-2 mencetak `**`. Baris ke-3, `***`, dan seterusnya.

Kode 6.1: Print 10 baris bintang menggunakan loop

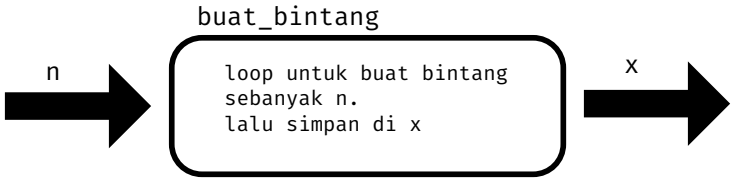
```
1 for i in range(1,11):
2     x = ""
3     for j in range(i):
4         x = x + "*"
5     print(x)
```

Baris 2, 3, dan 4 dalam kode tersebut ber*fungsi* untuk membuat string dengan karakter bintang sebanyak posisi baris dan menyimpannya di `x`. Kita bisa memisahkan 3 baris kode ini dan memberi nama sesuai tugasnya, yaitu membuat bintang sebanyak yang diinginkan. Perhatikan Kode 6.2.

Kode 6.2: Fungsi untuk membuat bintang

```
1 def buat_bintang(n):
2     x = ""
3     for i in range(n):
4         x = x + "*"
5     return x
```

Di Kode 6.2, baris 2, 3, dan 4 dari Kode 6.1 dimasukkan ke dalam suatu fungsi yang namanya `buat_bintang`. Fungsi dapat dianggap sebagai sebuah mesin yang menerima input, memprosesnya, dan mengeluarkan output (hasilnya). Dalam fungsi `buat_bintang`, inputnya adalah variabel `n`. Sementara, outputnya ada di baris 5, yaitu `x`. Variabel `x` berisi string bintang sebanyak `n`. Kata kunci `return` menunjukkan data (dalam hal ini `x`) yang dihasilkan fungsi `buat_bintang`. Perhatikan Gambar 6.1



Gambar 6.1: Fungsi `buat_bintang` menerima `n` dan mengeluarkan `x`

Dengan fungsi `buat_bintang` kita bisa membuat program di Kode 6.1 menjadi tampak lebih sederhana dan rapi serta bersih. Perhatikan Kode 6.3 terutama baris 7-9.

Kode 6.3: Kode 6.1 menjadi lebih mudah dibaca dengan fungsi

```
1 def buat_bintang(n):
2     x = ""
3     for i in range(n):
4         x = x + "*"
5     return x
6
7 for i in range(1,11):
8     bintang = buat_bintang(i)
9     print (bintang)
```

Kode 6.4: Kerjakan

```
1 def buat_bintang(n):
2     x = ""
3     for i in range(n):
4         x = x + "*"
5     return x
6
7 # printlah bintang sebanyak 5 bintang
8 # dengan menggunakan fungsi buat_bintang
```

Kita bisa menyimpan fungsi `buat_bintang` di file terpisah sehingga fungsi ini dapat digunakan oleh kode lain juga. Jika demikian, fungsi `buat_bintang`

harus diimpor terlebih dahulu. Misalkan kita simpan fungsi `buat_bintang` di file yang namanya bintang.py. Perhatikan Kode 6.5.

Kode 6.5: Menggunakana `buat_bintang` dari file bintang.py

```
1 from bintang import buat_bintang
2
3 for i in range(1,11):
4     bintang = buat_bintang(i)
5     print (bintang)
```

Baris 1 Kode 6.5 menyatakan bahwa kita mengimport dari file `bintang.py` fungsi `buat_bintang`.

Kode 6.6: Kerjakan: Buat program yang hasilnya seperti berikut.

```
1 *
2 **
3 ***
4 ****
5 ***
6 **
7 *
```

Kita dapat menambah fitur ke fungsi `buat_bintang`, yaitu dengan menentukan karakter yang akan dicetak. Dengan demikian, kita ubah nama fungsinya menjadi `buat_n_karakter`. Perhatikan Kode 6.7.

Kode 6.7: Fungsi `buat_n_karakter`

```
1 def buat_n_karakter(karakter, n):
2     x = ""
3     for i in range(n):
4         x = x + karakter
5     return x
6
7 print (buat_n_karakter("*", 2))
8 print (buat_n_karakter("+", 4))
9 print (buat_n_karakter("%", 6))
```

Kode 6.8: Kerjakan

```
1 # Buat fungsi untuk menghitung kuadrat dari sebuah angka
2 def kuadrat(angka):
3     pass # Ganti dengan kode
4
5 a = kuadrat(4)
6 print(a) # hasilnya 16
7
8 b = kuadrat(5)
9 print(b) # hasilnya 25
```

6.2 File

Sebuah program, kadang butuh menyimpan data dalam bentuk file. Beberapa aktivitas terkait file di antaranya adalah cek keberadaan file, membuat/menghapus file, membuka/menutup file, membaca data file, dan menulis data ke file.

6.2.1 Cek File

Untuk menangani file, tentu kita harus memastikan bahwa file tersebut ada. Kode 6.9 menunjukkan kode untuk cek keberadaan `file.txt` di folder yang sama dengan folder kode python-nya.

Kode 6.9: Cek keberadaan file

```
1  import os
2
3  if os.path.exists("file.txt"):
4      print ("file.txt ada")
5  else:
6      print ("file.txt tidak ada")
```

6.2.2 Membuat/Menghapus File

Untuk membuat file, perhatikan Kode 6.10.

Kode 6.10: `buat-file.py`

```
1  import os
2
3  print ("sebelum perintah open")
4  if os.path.exists("file.txt"):
5      print ("file.txt ada")
6  else:
7      print ("file.txt tidak ada")
8
9  # Karakter 'x' dalam fungsi open berarti
10 # fungsi ini akan error jika file sudah ada.
11 # Supaya tidak error, gunakan 'w' (write)
12 f = open("file.txt","x")
13
14 print ("setelah perintah open")
15 if os.path.exists("file.txt"):
16     print ("file.txt ada")
17 else:
18     print ("file.txt tidak ada")
```

Untuk menghapus file, perhatikan Kode 6.11.

Kode 6.11: `hapus-file.py`

```
1 import os
2
3 if os.path.exists("file.txt"):
4     print ("file.txt ada dan akan dihapus")
5     os.remove("file.txt")
6 else:
7     print ("file.txt tidak ada")
```

6.2.3 Membuka/Menutup File

Untuk membuka sebuah file, kita tuliskan kode seperti yang ditampilkan di Kode 6.12.

Kode 6.12: `buka-file.py`

```
1 import os
2
3 namafile = "file.txt"
4
5 if os.path.exists(namafile):
6     f = open(namafile)
7     print (namafile, "ada dan telah dibuka.")
8 else:
9     print (namafile, "tidak ada")
```

Setelah suatu file dibuka, dan kita selesai bekerja dengan file tersebut (misalnya selesai menulis atau membaca file), ada baiknya kita menutup file tersebut agar dibersihkan dari memori. Cara menutup file adalah dengan fungsi `close`. Perhatikan Kode 6.13.

Kode 6.13: `tutup-file.py`

```
1 import os
2
3 namafile = "file.txt"
4
5 if os.path.exists(namafile):
6     f = open(namafile)
7     f.close() # menutup file
8 else:
9     print (namafile, "tidak ada")
```

Fungsi `open` digunakan baik untuk membuka file maupun untuk membuat file. Beberapa variasinya adalah seperti yang ditunjukkan di Kode 6.14.

Kode 6.14: Variasi penggunaan perintah `open`

```
namafile="file.txt"

# Digunakan untuk MEMBUKA dan MENULIS "file.txt".
# Jika "file.txt" tidak ada, file akan dibuat.
# Hati-hati!!! 'w' akan menimpa isi file sebelumnya.
# Jika ingin menambahkan baris, gunakan 'a' (append)
f = open(namafile,'w') # write (menimpa)
f = open(namafile,'a') # append (menambah)

# Digunakan untuk MEMBUKA dan MEMBACA "file.txt".
# Jika "file.txt" tidak ada, error.
# Perintah berikut sama dengan open(namafile)
f = open(namafile,'r')

# Digunakan untuk MEMBUAT "file.txt" ('x' berarti create).
# Jika "file.txt" ada, error.
f = open(namafile,'x')
```

6.2.4 Membaca File

Setelah dibuka, kita tentu ingin membaca isi file. Misalkan saya memiliki file dengan nama `file.txt` yang isinya seperti di Kode 6.15.

Kode 6.15: `file.txt`

```
ini adalah file
file ini berisi teks
```

Cara membuka dan membaca file tersebut adalah seperti yang ditunjukkan di Kode 6.16. Hasilnya dapat dilihat di Kode 6.17.

Kode 6.16: `baca-file.py`

```
import os

namafile = "file.txt"

if os.path.exists(namafile):
    f = open(namafile, 'r')
    text1 = f.readline()
    print (text1)
    text2 = f.readline()
    print (text2)
else:
    print (namafile, "tidak ada")
```

Kode 6.17: Hasil dari `baca-file.py`

```
ini adalah file

file ini isinya teks
```

Fungsi `readline` di Kode 6.16 membaca isi file setiap baris. Setelah dibaca, isi baris tersebut disimpan dalam variabel `text1`. Baris yang disimpan di `text1`, mengandung karakter *newline* (baris baru). Jika kita gunakan fungsi `print(text1)`, kita jadi memiliki 2 karakter *newline*, yaitu dari isi file yang dibaca dan dari fungsi `print`. Itulah kenapa hasil di Kode 6.17 terdapat baris kosong. Untuk menghindarinya, kita ingin menghilangkan karakter *newline* di setiap baris yang kita baca dari file. Caranya adalah dengan menggunakan fungsi `strip()`. Perhatikan Kode 6.18.

Kode 6.18: `baca-file-strip.py`

```
import os

namafile = "file.txt"

if os.path.exists(namafile):
    f = open(namafile, 'r')
    text1 = f.readline().strip()
    print (text1)
    text2 = f.readline().strip()
    print (text2)
else:
    print (namafile, "tidak ada")
```

Kode 6.18 tidaklah efisien karena kita terus-terusan menuliskan fungsi `readline` sebanyak baris yang ingin kita baca. Bagaimana jika kita ingin membaca semua baris dalam file tapi kita tidak tahu berapa baris ada di dalam file tersebut? Untuk itu, kita sebaiknya menggunakan perulangan seperti di Kode 6.19.

Kode 6.19: `baca-file-loop.py`

```
import os

namafile = "file.txt"

if os.path.exists(namafile):
    f = open(namafile, 'r')

    for text in f:
        print (text.strip())
else:
    print (namafile, "tidak ada")
```

Jika kita ingin membagi setiap baris yang dibaca dari suatu file ke dalam kata-kata. Kita bisa gunakan perintah `split`. Perhatikan Kode 6.20 dan hasilnya di Kode 6.21.

Kode 6.20: `split-baris.py`

```
1  import os
2
3  namafile = "file.txt"
4
5  if os.path.exists (namafile):
6      f = open(namafile,'r')
7      baris = f.readline().strip()
8      kata = baris.split()
9      print (kata)
10 else:
11     print (namafile, "tidak ada")
```

Kode 6.21: Hasil dari `split-baris.py`

```
1  ['ini', 'adalah', 'file']
```

6.2.5 Menulis File

Untuk menulis teks ke file, kita bisa menggunakan fungsi `open` dengan parameter `w` atau `a`. Gunakan `w` jika ingin menimpa isi file dan gunakan `a` jika ingin menambahkan baris ke file. Perhatikan Kode 6.22.

Kode 6.22: `tulis-file.py`

```
1  import os
2
3  namafile = "file2.txt"
4
5  if os.path.exists(namafile):
6      f = open(namafile, 'r')
7      print ("sebelum ditambahkan")
8      print ("--------------------------")
9      for x in f:
10         print (x.strip())
11     f.close()
12
13     f = open(namafile, 'a')
14     f.write ("ini adalah baris tambahan\n")
15     f.close()
16
17     print ("")
18     print ("setelah ditambahkan")
```

```
19      print ("-----------------------------")
20      f = open(namafile, 'r')
21      for x in f:
22          print (x.strip())
23      f.close()
24  else:
25      print (namafile, "tidak ada")
```

6.3 Kerjakan

1. Buatlah fungsi yang menerima karakter dan jumlah karakter serta mengembalikan string yang berisi rentetan karakter sebanyak jumlah karakter yang dimasukkan.

Kode 6.23: Kerangka kode untuk soal no. 1

```
1  def nkarakter(karakter, jumlah):
2      # tulis kode di sini
3
4  # contoh penggunaan
5  x = nkarakter("#",5)
6
7  # perintah ini mencetak ##### ke layar
8  print(x)
```

2. Buatlah fungsi yang menerima string lalu mengembalikan jumlah karakter dalam string tersebut.

Kode 6.24: Kerangka kode untuk soal no. 2

```
1  def jumlah_karakter(input_string):
2      n_karakter = 0
3      # tulis kode di sini
4
5      return n_karakter
6
7  # contoh penggunaan
8  x = jumlah_karakter("multimedia")
9
10 # perintah ini harus mencetak angka 10 ke layar
11 print(x)
```

3. Sebuah robot berada di koordinat (0,0). Buatlah 4 fungsi, yaitu `maju()` untuk menambah `x` satu satuan, `mundur()` untuk mengurangi `x` satu satuan, `belok_kanan()` untuk menambah `y` satu satuan, dan `belok_kiri()`

untuk mengurangi `y` satu satuan. Perhatikan kerangka program di Kode 6.25.

Kode 6.25: Kerangka kode untuk soal no. 3

```
1  def maju(posisi):
2      # kode di sini
3
4  def mundur(posisi):
5      # kode di sini
6
7  def belok_kanan(posisi):
8      # kode di sini
9
10 def belok_kiri(posisi):
11     # kode di sini
12
13 selesai = False
14 posisi_robot = (0,0) # tuple
15 while selesai == False:
16     print(" --------------- ")
17     print("Pilih perintah untuk robot.")
18     print("1. maju")
19     print("2. mundur")
20     print("3. belok kanan")
21     print("4. belok kiri")
22     print("5. selesai")
23     print(" --------------- ")
24     perintah = input("Masukkan perintah ")
25     perintah = int(perintah)
26
27     if perintah == 1:
28         posisi_robot = maju(posisi_robot)
29         print(posisi_robot)
30     elif perintah == 2:
31         posisi_robot = mundur(posisi_robot)
32         print(posisi_robot)
33     elif perintah == 3:
34         posisi_robot = belok_kanan(posisi_robot)
35         print(posisi_robot)
36     elif perintah == 4:
37         posisi_robot = belok_kanan(posisi_robot)
38         print(posisi_robot)
39     elif perintah == 5:
40         selesai = True
41         print(posisi_robot)
```

4. Ubahlah Kode 6.25. Jadikan 4 fungsi `maju()`, `mundur()`, `belok_kanan()`, dan `belok_kiri()` ke dalam satu fungsi. Akan tetapi, 1 fungsi ini

menerima masukan tambahan berupa petunjuk apakah robot harus maju, mundur, belok kanan, atau belok kiri.

5. Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri. Contohnya adalah fungsi di Kode 6.26. Menurut anda, apa yang akan terjadi jika Kode 6.26 dijalankan?

Kode 6.26: Fungsi rekursif

```
1 def rekursif():
2     print("x")
3     rekursif()
4
5 rekursif()
```

6. Buat program untuk menyimpan data nama dan nilai. Simpan data nama dan nilai tersebut di dalam file.

7. Buat program kalkulator dengan menu seperti berikut.

   (a) Tambah
   (b) Kurang
   (c) Kali
   (d) Bagi
   (e) Keluar

8. Buat program tebak kata. Baca daftar kata yang ada di dalam file (buat dulu file-nya, misal 20 kata. Tulis setiap kata per baris dalam file tersebut). Lalu baca filenya. Simpan kata-kata tersebut ke dalam `list`. Lalu buat komputer untuk memilih secara acak satu kata. User hanya dapat menebak huruf-huruf yang ada dalam kata tersebut 10 kali. Setiap kali satu huruf berhasil ditebak, tampilkan kata dengan huruf-huruf yang sudah ditebak dalam posisi yang benar.

Modul 7

Tipe Data Kustom

1. **Tujuan**: Mengenal tipe data kustom di Python
2. **Pengerjaan**: Perorangan
3. **Durasi**: 4 × 50 menit
4. **Perangkat**: Laptop dan Internet

Kita sudah belajar 4 tipe data dasar dalam Python, yaitu integer, string, float, dan boolean. Selain itu, ada juga 3 tipe data lanjutan, yaitu list, tuple, dan dictionary.

Sebagai programer, kita sebenarnya bisa membuat tipe data sendiri. Misalnya ketika membuat program untuk pengelolaan data mahasiswa, daripada menggunakan tipe data dasar untuk menampung data nama, nim, umur, gender, dan alamat, kita dapat menggabungkan data tersebut ke dalam satu tipe data kustom yang kita bikin sendiri, misalnya yang kita sebut `biodata`. Perhatikan Kode 7.1.

Kode 7.1: Cara mendefinisikan tipe data baru

```
1  class biodata:
2      def __init__(self, nama, nim, umur, gender, alamat):
3          self.nama = nama
4          self.nim = nim
5          self.umur = umur
6          self.gender = gender
7          self.alamat = alamat
8
9  anggota1 = biodata("budi", 1, 18, 'L', 'bandung')
10 anggota2 = biodata("wati", 2, 19, 'P', 'jakarta')
11
12 print(anggota1.nama, anggota1.umur)
13 print(anggota2.nama, anggota2.umur)
```

Fungsi `__init__(self, nama, nim, umur, gender, alamat)` disebut sebagai konstruktor. Fungsi ini dipanggil setiap kali data dengan tipe `biodata` dibuat, contohnya di baris 9 dan 10 dalam Kode 7.1.

Setiap data bertipe `biodata` dibuat, kita menamakannya sebagai proses instansiasi (*instantiation*). Jadi, `anggota1` dan `anggota2` adalah hasil instansiasi dari tipe data (*class*) `biodata`. Variabel `self` adalah variabel yang hanya dapat diakses di dalam kelas dan merujuk ke setiap hasil instansiasi. Contohnya, `self` di `anggota1` mengacu ke `anggota1` itu sendiri dan bukan ke `anggota2`.

Kode 7.2: Mengubah nama dan umur

```
1  class biodata:
2      def __init__(self, nama, nim, umur, gender, alamat):
3          self.nama = nama
4          self.nim = nim
5          self.umur = umur
6          self.gender = gender
7          self.alamat = alamat
8
9  anggota1 = biodata("budi", 1, 18, 'L', 'bandung')
10 print(anggota1.nama, anggota1.umur)
11
12 anggota1.nama = "amir"
13 anggota1.umur = 19
14
15 print(anggota1.nama, anggota1.umur)
```

Selain fungsi konstruktor, ke dalam `class` juga dapat kita tambahkan fungsi-fungsi lain. Misalnya kita ingin menghitung tahun kelahiran seseorang dari umurnya. Untuk itu kita buat fungsi `hitung_tahun_lahir` di dalam kelas `biodata` Untuk menghitung tahun lahir, kita perlu memasukkan data tahun sekarang ke dalam fungsi. Perhatikan Kode 7.3.

Kode 7.3: Menambahkan fungsi `hitung_tahun_lahir` di `class biodata`.

```
1  class biodata:
2      def __init__(self, nama, nim, umur, gender, alamat):
3          self.nama = nama
4          self.nim = nim
5          self.umur = umur
6          self.gender = gender
7          self.alamat = alamat
8
9      def hitung_tahun_lahir(self, tahun_sekarang):
10         tahun_lahir = tahun_sekarang - self.umur
11         return tahun_lahir
12
```

```
13 anggota1 = biodata("budi", 1, 18, 'L', 'bandung')
14
15 lahir = anggota1.hitung_tahun_lahir (2024)
16
17 print("tahun lahir", anggota1.nama, "adalah", lahir)
```

7.1 Contoh Program

Misalkan kita akan membuat game pertarungan antara jago bela diri. Pertama, kita bisa membuat tipe data baru berupa petarung. Mari kita namakan tipe data ini `Petarung`. Apa sajakah yang dimiliki oleh seorang petarung? Perhatikan Kode 7.4 dan hasilnya di Kode 7.5

Kode 7.4: Tipe data `Petarung`

```
1  import random
2
3  class Petarung:
4      def __init__(self,nama):
5          self.nama = nama
6          self.kesehatan = 100
7          self.kekuatan = random.randint(0,100)
8          self.kecerdasan = random.randint(0,100)
9          self.kelincahan = random.randint(0,100)
10         self.senjata = random.randint(0,100)
11
12     def print_data(self):
13         print(self.nama)
14         print("Kesehatan = ", self.kesehatan)
15         print("Kekuatan = ", self.kekuatan)
16         print("Kecerdasan = ", self.kecerdasan)
17         print("Kelincahan = ", self.kelincahan)
18         print("Senjata = ", self.senjata)
19
20 naruto = Petarung("Naruto")
21 naruto.print_data()
```

Kode 7.5: Naruto adalah instansiasi dari `Petarung`

```
1 Naruto
2 Kesehatan =  100
3 Kekuatan =  98
4 Kecerdasan =  94
5 Kelincahan =  51
6 Senjata =  45
```

Mari sekarang kita bikin Sasuke.

Kode 7.6: Instansiasi `Petarung` ke 2 objek

```
# saya pindahkan class Petarung ke file petarung.py
# lalu saya import
from petarung import Petarung

naruto = Petarung("Naruto")
sasuke = Petarung("Sasuke")

naruto.print_data()
print("----------------------")
sasuke.print_data()
```

Kode 7.7: Hasil Kode 7.6

```
Naruto
Kesehatan = 100
Kekuatan = 62
Kecerdasan = 94
Kelincahan = 13
Senjata = 44
----------------------
Sasuke
Kesehatan = 100
Kekuatan = 66
Kecerdasan = 49
Kelincahan = 44
Senjata = 51
```

Sekarang, mari kita tandingkan antara Naruto dan Sasuke. Untuk itu, kita buat tipe data `Tanding` seperti di Kode 7.8 dan kita tandingkan seperti di Kode 7.9. Hasilnya dapat dilihat di Kode 7.10.

Kode 7.8: Tipe data `Tanding`

```
class Tanding:
    def __init__(self, petarung1, petarung2):
        self.petarung1 = petarung1
        self.petarung2 = petarung2
        self.pemenang = ""

    def mulai(self):
        """
        pemenang pertarungan ditentukan dengan cara sbb.
        setiap sifat antara 2 petarung dibandingkan
        yang lebih besar mendapatkan 1 point.
        pemenang adalah petarung yang memiliki
        point paling besar.
        """
        nilai_petarung1 = 0
```

```
16          nilai_petarung2 = 0
17
18          if (self.petarung1.kekuatan > self.petarung2.kekuatan):
19              nilai_petarung1 = nilai_petarung1 + 1
20          elif (self.petarung1.kekuatan < self.petarung2.kekuatan):
21              nilai_petarung2 = nilai_petarung2 + 1
22
23          # dan seterusnya untuk kecerdasan, kelincahan,
24          # dan juga senjata
25
26          if (nilai_petarung1 > nilai_petarung2):
27              self.pemenang = self.petarung1.nama
28          elif (nilai_petarung1 < nilai_petarung2):
29              self.pemenang = self.petarung2.nama
30          else:
31              self.pemenang = ""
32
33      def print_pemenang(self):
34          if (self.pemenang != ""):
35              print("Pemenangnya: " + self.pemenang)
36          else:
37              print("Seri!!")
```

Kode 7.9: Naruto dan Sasuke bertanding.

```
1  from petarung import Petarung
2  from tanding import Tanding
3
4  naruto = Petarung("Naruto")
5  sasuke = Petarung("Sasuke")
6  berantem = Tanding(naruto, sasuke)
7  berantem.mulai()
8
9  print("--------------------")
10 naruto.print_data()
11 print("--------------------")
12 sasuke.print_data()
13 print("--------------------")
14 berantem.print_pemenang()
15 print("--------------------")
```

Kode 7.10: Hasil pertandingan

```
1  --------------------
2  Naruto
3  Kesehatan = 100
4  Kekuatan = 58
5  Kecerdasan = 43
6  Kelincahan = 95
```

```
7  Senjata = 70
8  --------------------
9  Sasuke
10 Kesehatan = 100
11 Kekuatan = 28
12 Kecerdasan = 20
13 Kelincahan = 16
14 Senjata = 92
15 --------------------
16 Pemenangnya: Naruto
17 --------------------
```

7.2 Kerjakan

1. Lengkapi Kode 7.8 dengan bagian perhitungan untuk kecerdasan, kelincahan, dan juga senjata
2. Lengkapi setiap petarung dengan data skor
3. Buatlah program pertarungan ini agar dilakukan berulang-ulang sampai pemain memutuskan untuk keluar program. Tambahkan menu seperti berikut.

   (a) Lihat petarung
   (b) Edit petarung
   (c) Tambah petarung
   (d) Hapus petarung
   (e) Bertanding
   (f) Keluar

   Untuk menu `Lihat petarung`, tampilkan datanya juga. Selain itu, simpan data setiap petarung dalam file (Gunakan 1 file untuk setiap petarung). Untuk menu `Bertanding`, ketika dipilih menu dilanjutkan dengan pemilihan 2 nama petarung.
4. Tambahkan data jumlah kalah, menang, dan seri untuk setiap petarung
5. Buatlah agar kesehatan setiap petarung yang kalah dikurangi 5 poin
6. Buatlah agar setiap kali seorang petarung menang, kekuatannya bertambah secara acak antara 1-5 poin.
7. Buatlah satu fungsi di *class* `Petarung` untuk men-generate ulang nilai-nilai kekuatan, kecerdasan, kelincahan, dan senjatanya namun dengan biaya kesehatannya dikurangi 10 poin.

Modul 8

Debugging dan Versioning

1. **Tujuan**: Mengenal debugging dan manajemen kode lewat versioning
2. **Pengerjaan**: Perorangan
3. **Durasi**: 4 × 50 menit
4. **Perangkat**: Laptop dan Internet

8.1 Debugging

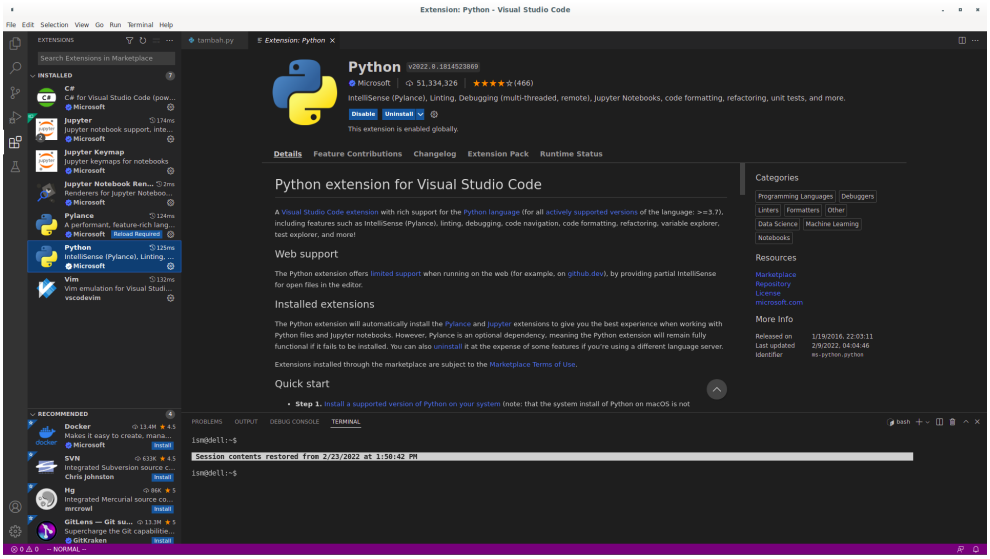
Sebuah program bisa jadi sangat kompleks. Program seperti Microsoft Windows, atau games seperti FIFA, terdiri dari puluhan juta baris perintah. Program seperti itu dikerjakan oleh puluhan orang yang menggarap masing-masing bagian program. Dengan program sekompleks itu, bisa dipastikan akan terdapat kesalahan dalam program. Kesalahan dalam program biasanya disebut *bug*. Cara mencari kesalahan dalam program disebut *debugging*.

Kita hanya akan mengenal dasar-dasar *debugging* saja. Tidak akan sampai detail mendalami teknik debugging. Tools yang digunakan seharusnya sudah ter-*install* di VSCode. Jika belum, *install tool* untuk *debugging* Python di VSCode. Perhatikan Gambar 8.1.

Salah satu yang dilakukan saat debugging adalah, kita menentukan *breakpoint*, yaitu titik tempat program harus berhenti agar kita dapat memeriksa isi dari variabel-variabel yang ada di dalam program saat itu. Sebagai contoh, tulislah Kode 8.1 lalu jalankan.

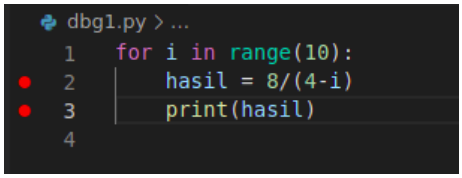
Kode 8.1: Program untuk di-*debug*

```
1 for i in range(10):
2     if (i != 4):
3         hasil = 8/(4-i)
4     print(hasil)
```



Gambar 8.1: Python debugger di VSCode

Program di Kode 8.1 akan mengalami *error* karena saat `i = 4` program akan melakukan operasi pembagian dengan bilangan 0 yang tidak diperbolehkan. Akan tetapi, meskipun kita sudah tahu sumber kesalahan, mari kita lihat program ini langkah per langkah saat dijalankan. Untuk itu, klik di sebelah kiri angka 2 (baris 2) dan di baris ke-3 sehingga muncul titik warna merah seperti terlihat di Gambar 8.2. Titik-titik ini disebut *breakpoint*.



Gambar 8.2: Breakpoint di baris ke-2 dan ke-3

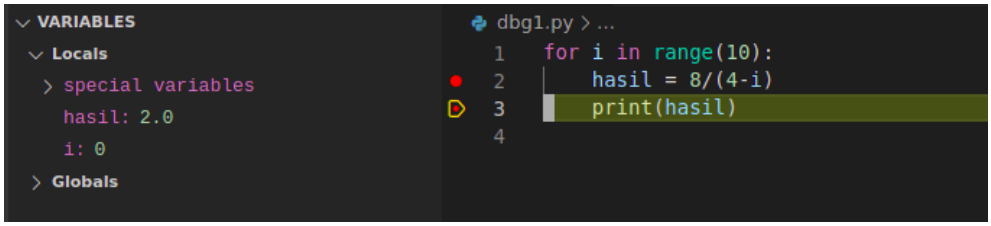
Setelah menyimpan *breakpoint* di baris ke-2 dan ke-3, debug lah program dengan menekan `F5`. Program akan berjalan dan berhenti di baris ke-2. Perhatikan informasi di panel sebelah kiri (Gambar 8.3).

Tekan lagi `F5` dan program akan berjalan lagi. Akan tetapi, karena ada breakpoint di baris ke-3, program kembali berhenti. Perhatikan Gambar 8.4.

Jika ditekan lagi `F5`, program akan loop dan kembali ke baris ke-2. Di baris ke-2 program berhenti kembali. (Gambar 8.5).

Jika diteruskan, akhirnya pada suatu saat program akan error dan muncul pesan seperti dalam Gambar 8.6. Di panel sebelah kiri terlihat bahwa pada

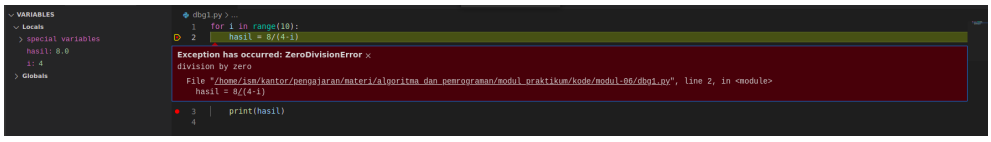
Gambar 8.3: Program berhenti di baris ke-2. Sebelum baris kedua dijalankan, variabel `i` nilainya 0



Gambar 8.4: Sekarang program berhenti di baris ke-3. Sebelum baris ketiga dijalankan, variabel `i` nilainya 0 dan variabel `hasil` nilainya 2.0

Gambar 8.5: Program berhenti lagi di baris ke-2. Variabel `i = 1` dan `hasil = 2.0`

saat error, nilai `i = 4`. Dari sini sebagai programmer kita harus tahu bahwa perintah di baris ke-2, akan menghasilkan error saat `i = 4`.



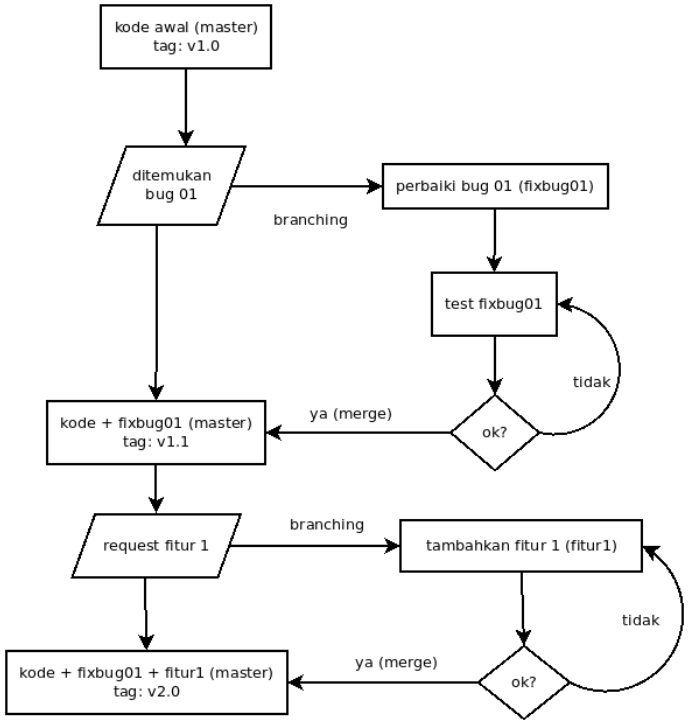
Gambar 8.6: Program error saat `i = 4`

Secara simpel, itulah yang dilakukan saat *debugging*. Tentu saja dalam praktiknya banyak hal-hal lain. Akan tetapi, untuk saat ini, cukup itu saja untuk bekal mendalami lebih jauh lagi proses debugging saat kita sudah menjadi programmer yang berurusan dengan jutaan baris kode.

8.2 Versioning

Selain *debugging*, proses lain yang tidak kalah penting saat membuat program adalah manajemen kode. Sebuah program yang berkembang dari nol hingga ratusan baris lalu jutaan dan puluhan juta baris, tentu harus dikelola kodenya. Apalagi jika banyak orang yang terlibat dalam pembuatan program tersebut.

Salah satu perangkat lunak untuk manajemen kode adalah git. Dengan git, kita bisa melacak setiap perubahan dalam kode. Dengan demikian, kita mudah untuk mengelola kode. Biasanya, alur kerja manajemen kode menggunakan git diperlihatkan di Gambar 8.7.



Gambar 8.7: Alur manajemen kode menggunakan git

Jika diuraikan, alur kerja manajemen kode di Git berdasarkan Gambar 8.7 adalah sebagai berikut.

1. Misalkan kita sudah punya kode program dengan fitur minimal. Kita release program ini dengan kode versi 1.0 (tag v1.0). Kita akan mulai manajemen kode dari kode versi 1.0 ini. Kode 1.0 ini kita simpan di cabang utama (*branch*) git, yaitu master.

2. Suatu saat, di kode 1.0 ditemukan bug. Untuk memperbaikinya, kita buat *branch* baru, misalnya namanya fixbug01. Gunanya membuat *branch* ini adalah agar kode 1.0 kita tidak diutak-atik.

3. Di *branch* baru ini, kita perbaiki kodenya. Setelah kita uji dan merasa bahwa bug yang ditemukan sudah berhasil diperbaiki, kita simpan kode hasil perbaikan di git. Proses ini namanya *commit*.

4. Setelah perbaikan bug dianggap selesai, saatnya menggabungkan kode hasil perbaikan di *branch* fixbug01 ke *branch* utama, yaitu branch *master*. Proses ini namanya *merge*.

5. Setelah proses *merge* berhasil, kita bisa menghapus *branch* fixbug01 dan melabeli kode terbaru kita dengan versi 1.1 (tag v1.1).

6. Proses yang sama berlaku jika kita ingin menambahkan fitur ke kode utama (*master*).

7. Dengan menggunakan Git, setiap penambahan/perubahan kode dapat dilacak dan tidak mengganggu kode yang sudah di-release. Selain itu, jika program dibuat oleh tim, masing-masing tim dapat membuat *branch* sendiri untuk memperbaiki bug atau menambah fitur. Setelah selesai, masing-masing dapat melakukan *merge* ke *branch* master.

8.2.1 Instalasi Git

Untuk instalasi Git di Windows, lakukan langkah-langkah berikut.

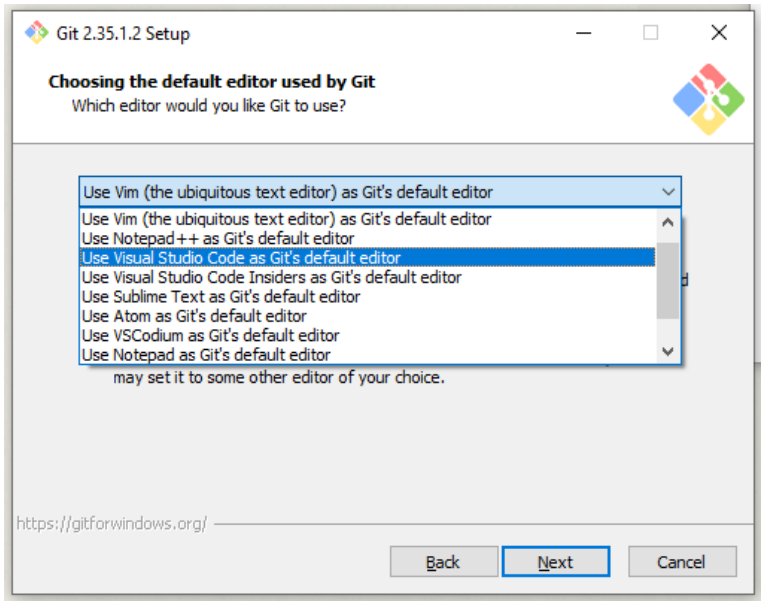
1. Kunjungi `https://git-scm.com/download/win` dan pilih file yang sesuai dengan arsitektur komputer kalian (32 bit atau 64 bit).

2. Setelah selesai download, klik dua kali file tersebut. Akan muncul dialog seperti pada Gambar 8.8.

3. Klik `Next` hingga ke dialog di Gambar 8.9. Piilh Visual Code Studio sebagai default editor bagi Git.

4. Selanjutnya, klik `Next` terus sampai muncul klik `Install`. Setelah klik `Install`, tunggu beberapa saat sampai instalasi selesai.

Gambar 8.8: Dialog awal instalasi Git



Gambar 8.9: Pilih VSCode sebagai Git default editor

8.2.2 Inisialisasi Git

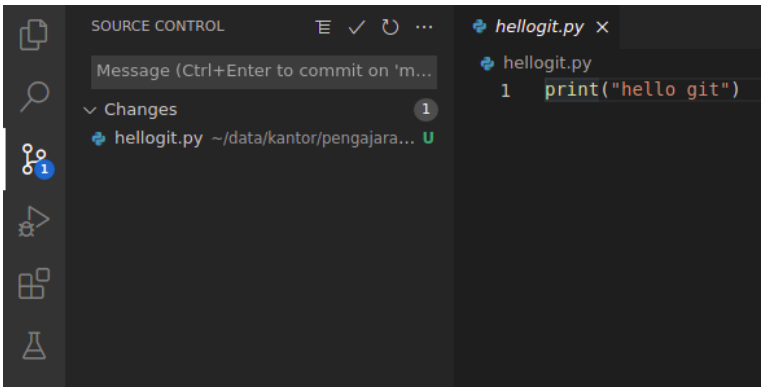
Misalkan kita membuat program hellogit. File program kita ada di folder `myprogram`. Untuk manajemen kode, kita jadikan folder `myprogram` ini seba-

gai repositori git. Caranya adalah, buka folder `myprogram` di VSCode (`File` → `Open Folder`).

Selanjutnya, tambahkan 1 file python dengan nama file `hellogit.py`. Isi file dengan 1 baris: `print(''hello git'')`. Kemudian, klik ikon Source Control (`Ctrl + Shift + G`) lalu klik `Initialize Repository`. Perhatikan Gambar 8.10.

Gambar 8.10: Inisialisasi repositori git.

Setelah menginisialisasi repositori, panel source control akan berubah seperti dalam Gambar 8.11.



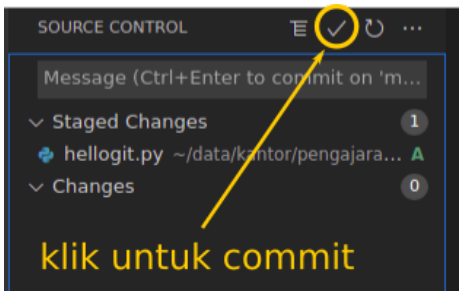
Gambar 8.11: Tampilan panel source control setelah inisialisasi git.

8.2.3 Melacak Perubahan Kode (`add` dan `commit`)

Setiap kali kita membuat file baru atau melakukan perubahan di file yang ada di repositori git, kita menyuruh git untuk melacaknya. Di Gambar 8.11

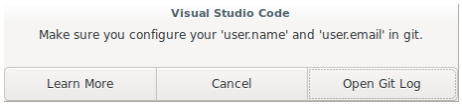
terlihat file baru kita, yaitu `hellogit.py`, memiliki huruf `U` di sampingnya. Ini menandakan bahwa file tersebut belum terlacak (*Untracked*) oleh git. Untuk melacaknya, arahkan kursor ke arah nama file tersebut sehingga muncul simbol plus (+). Klik simbol + tersebut untuk melacak file. Setelah ditambahkan, huruf `U` akan berubah menjadi huruf `A` (*Added*).

Selanjutnya, jika perubahan yang dilakukan dirasakan cukup kita harus menandai dan menyuruh git untuk mencatat perubahan ini. Caranya adalah dengan melakukan `commit`. Setiap commit dibarengi dengan pesan terkait perubahan yang terjadi. Dengan cara ini, setiap perubahan dalam kode kita bisa terlacak. Jika pada suatu saat kita ingin mengembalikan kode kita ke sebelum perubahan terjadi, kita dapat dengan mudah melakukannya. Untuk `commit`, klik simbol centang di bagian atas panel. Lalu, isikan pesan `commit`-nya. Perhatikan Gambar 8.12.



Gambar 8.12: Simbol untuk melakukan commit. Setelah diklik, isikan pesan `commit`.

Jika saat `commit` muncul *error* seperti dalam Gambar 8.13, itu berarti kita belum menentukan user dan email untuk git.



Gambar 8.13: Error saat melakukan `commit`

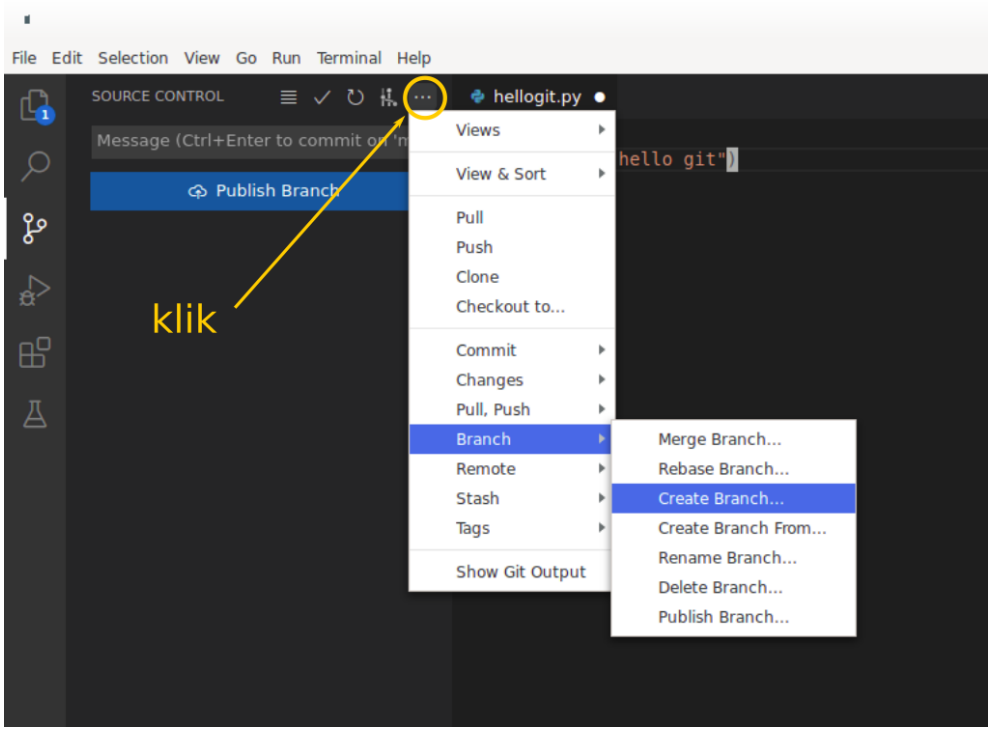
Untuk menentukan user dan email user git, bukalah folder `myprogram` di Windows Explorer, lalu klik kanan di tempat kosong dan klik Git Bash. Ketiklah dua perintah seperti di Kode 8.2.

Kode 8.2: Menentukan user dan email git

```
1 git config user.name "budi"
2 git config user.email "budi@email.com"
```

8.2.4 `Branch`

Seperti yang dapat dilihat di Gambar 8.7, alur penggunaan git adalah dengan membuat `branch` setiap kali kita ingin melakukan perubahan atau penambahan kode. Setelah perubahan selesai dilakukan, kita dapat melakukan `merge` dengan `branch` master. Untuk membuat `branch`, klik titik tiga di source control panel, lalu pilih `Branch` → `Create Branch...` (Gambar 8.14). Selanjutnya, beri nama `branch`.



Gambar 8.14: Membuat `branch`

Setelah membuat `branch`, kita otomatis berada di `branch` baru tersebut. Silakan lakukan perbaikan terhadap kode. Setelah selesai, lakukan `commit`. Jika dirasa pekerjaan di `branch` ini selesai, gabungkan perubahan di `branch` ini dengan `branch` master dengan melakukan `merge`.

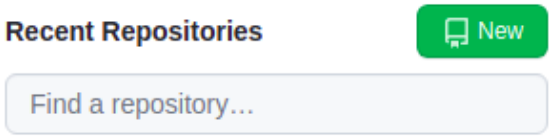
Cara melakukan merge adalah dengan pindah terlebih dahulu ke `branch` master. Caranya adalah dengan klik nama `branch` sekarang di kiri bawah VSCode lalu pilih `branch` mater. Lalu, klik lagi titik tiga (panel source control) dan pilih `Branch` → `Merge Branch...`. Pilih `branch` yang akan di-merge. Setelah selesai melakukan `Merge`, hapuslah `branch` tempat kita melakukan perbaikan tadi dengan memilih `Branch` → `Delete Branch...`.

Salah satu fitur yang berguna dari git adalah memberikan tag (label) untuk kode saat ini. Misalkan kita telah selesai memperbaiki satu bug, kita dapat memberikan tag sebelum kita melanjutkan perubahan lain pada kode. Dengan tag, kita dapat kembali ke posisi kode saat diberikan tag. Cara memberikan tag adalah dengan memilih titik tiga, lalu `Tags` → `Create Tag`.

8.2.5 GitHub

Menggunakan git, tim dapat bekerja membangun program dengan alur kerja yang tidak saling mengganggu. Setiap anggota tim dapat mengerjakan bagian dari program dan masing-masing membuat branch sendiri. Setelah selesai, masing-masing dapat merge branch mereka sehingga tidak saling mengganggu. Untuk itu, diperlukan satu server git. Salah satu server git yang bersifat publik adalah GitHub (`https://github.com`).

Untuk menggunakan git, kode program yang kita buat harus berada dalam satu folder yang kita sebut repositori git. Untuk membuat repositori git di Github, klik tombol `New`. Perhatikan Gambar 8.15.

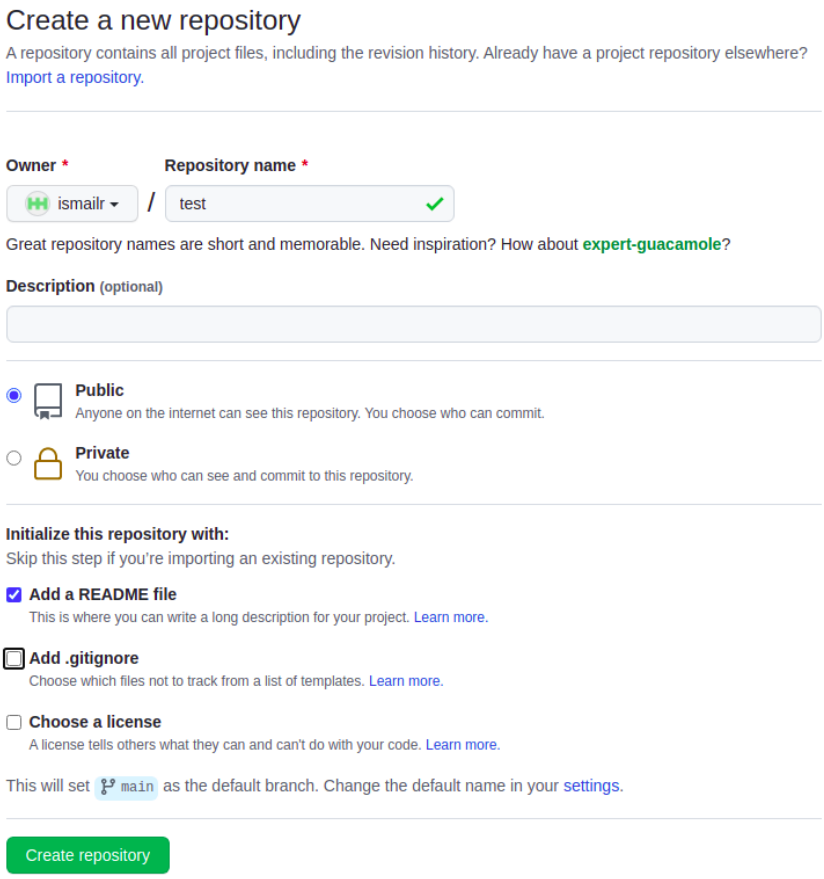


Gambar 8.15: Membuat repositori git baru di GitHub.

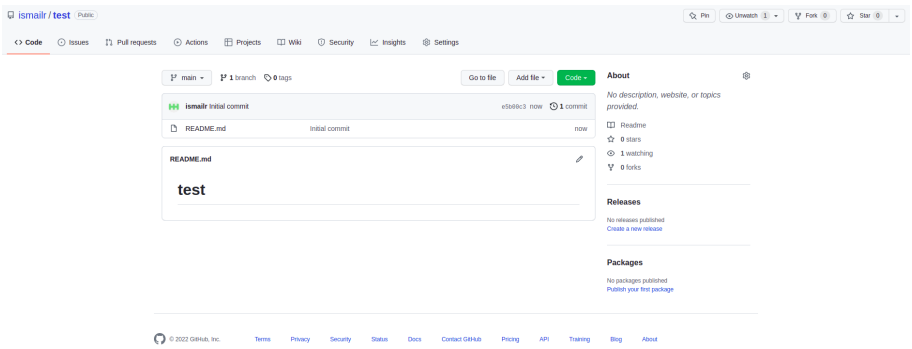
Setelah menekan tombol `New`, isilah form pembuatan repositori baru di GitHub seperti di Gambar 8.16. Setelah selesai, klik `Create repository`. Tampilan repositori yang sudah dibuat dapat dilihat di Gambar 8.17.

Repositori git yang sudah dibuat di GitHub dapat kita clone ke komputer lokal. Untuk apa? Agar kita dapat mengerjakan kodenya di komputer lokal. Setiap kali kita selesai update kode, kita dapat `commit` di server lokal, lalu upload (istilah git-nya `push`) ke server GitHub. Dengan demikian, kode kita aman di server GitHub dan jika ada yang akan berkontribusi dalam pembuatan program, juga dapat bekerja di repositori yang sama.

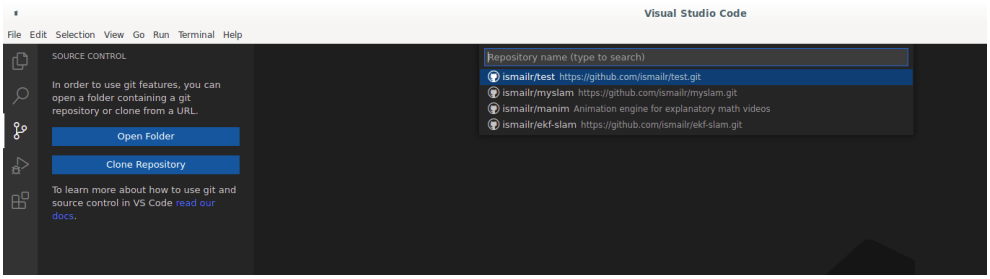
Untuk `clone` repositori di GitHub, terutama ke VSCode, kita dapat klik tombol `Clone Repository` di VSCode dan lakukan login. Saat berhasil, berbagai repositori kita di GitHub akan tempil dan kita dapat memilih repositori yang akan kita clone. Perhatikan Gambar 8.18.



Gambar 8.16: Isilah form untuk membuat repositori baru di GitHub.



Gambar 8.17: Repositori yang sudah selesai dibuat di GitHub.



Gambar 8.18: Memilih repositori di GitHub untuk di-`clone` ke komputer lokal.

8.3 Kerjakan

1. Buatlah 1 repositori baru di GitHub.
2. Minta 1 teman untuk fork repositori yang baru dibuat tersebut.

Modul 9

Asesmen dan Proyek

9.1 Asesmen

Dalam asesmen, mahasiswa diminta membuat program sederhana menggunakan 3 fitur dalam Python, yaitu variabel, percabangan, dan pengulangan. Asesmen dilakukan secara onsite dan bersifat *live coding*. Artinya mahasiswa diminta membuat program langsung di tempat setelah soal diberikan. Waktu yang diberikan ±5 menit. Contoh soalnya adalah sebagai berikut.

1. Buatlah program yang ketika dijalankan, mencetak 10 baris di terminal.
   - Setiap baris dimulai dengan nomor baris (dimulai dari 1).
   - Jika baris ganjil: tertulis "ganjil"
   - Jika baris genap: tertulis "genap"
   - Perhatikan hasil program di Kode 9.1.

Kode 9.1: Contoh keluaran

```
1 ganjil
2 genap
3 ganjil
4 genap
5 ganjil
6 genap
7 ganjil
8 genap
9 ganjil
10 genap
```

2. Buatlah program yang ketika dijalankan, mencetak 10 baris di terminal.
   - Setiap baris dimulai dengan nomor baris (dimulai dari 1).
   - Jika nomor baris kurang dari 6, tertulis karakter "*" sebanyak barisnya.

- Jika nomor baris lebih dari 5, tertulis karakter "+" sebanyak barisnya.
- Perhatikan hasil program di Kode 9.2.

Kode 9.2: Contoh keluaran

```
1 *
2 **
3 ***
4 ****
5 *****
6 ++++++
7 +++++++
8 ++++++++
9 +++++++++
10 ++++++++++
```

3. Buatlah program yang ketika dijalankan, mencetak 20 baris di terminal.
   - Setiap baris dimulai dengan nomor baris (dimulai dari 1).
   - Jika nomor baris kelipatan 2, print "kelipatan 2"
   - Jika nomor baris kelipatan 3, print "kelipatan 3"
   - Jika nomor baris kelipatan 2 dan 3, print "kelipatan 2 dan 3"
   - Lainnya, print bintang sebanyak nomor barisnya
   - Perhatikan hasil program di Kode 9.3.

Kode 9.3: Contoh keluaran

```
1 *
2 kelipatan 2
3 kelipatan 3
4 kelipatan 2
5 *****
6 kelipatan 2 dan 3
7 *******
8 kelipatan 2
9 kelipatan 3
10 kelipatan 2
11 ***********
12 kelipatan 2 dan 3
13 *************
14 kelipatan 2
15 kelipatan 3
16 kelipatan 2
17 *****************
18 kelipatan 2 dan 3
19 *******************
20 kelipatan 2
```

9.2 Proyek 1

Untuk proyek 1, mahasiswa diminta untuk membuat sebuah program sederhana berbasis teks. Spesifikasi programnya adalah sebagai berikut.

1. Program memiliki menu (jumlah pilihan dalam menu bebas selama terdapat pilihan keluar (`exit`)). Ini artinya, program harus berjalan terus-menerus sampai user memutuskan untuk keluar.
2. Program harus ditulis menggunakan kelas (`class`) dan juga fungsi.
3. Program harus menyimpan sesuatu ke file sehingga data milik user tidak hilang ketika keluar dari program.

Proyek ini dikerjakan secara berkelompok (maksimal 3 orang) dengan pembagian tugas harus jelas dan semua anggota harus ngoding. Pengujian program (asesmen 2) dilakukan perseorangan. Contoh program dapat dilihat di https://github.com/ismailr/alpro (contoh ini belum selesai dan belum menyertakan fitur baca/tulis ke file tapi sudah dapat dijalankan).

9.3 Proyek 2

Untuk proyek 2, mahasiswa diminta membuat video animasi yang dibuat menggunakan python. Video dibuat dengan bantuan pustaka manim (https://www.manim.community/).

Gambar 9.1: Logo Manim

Manim adalah pustaka yang awalnya digunakan untuk membuat *mathematical animations*. Akan tetapi, manim tidak hanya dapat digunakan untuk membuat animasi matematika. Manim menyediakan fitur untuk membuat objek geometri, menganimasikan pergerakannya dan juga perubahan bentuknya. Manim juga dapat memanfaatkan gambar sebagai objek. Untuk lebih lengkapnya, dapat dilihat dokumentasi dari manim (Ini adalah asesmen 3 sehingga diharapkan mahasiswa yang sampai tahap ini mampu menggunakan pustaka *open source*, meng-installnya serta membaca dokumentasinya).

9.4 Contoh Program untuk Asesmen

1. Buatlah program yang ketika dijalankan, mencetak 10 baris di terminal (Kode 9.4).
   - Setiap baris dimulai dengan nomor baris (dimulai dari 1).
   - Jika baris ganjil: tertulis "ganjil"
   - Jika baris genap: tertulis "genap"
   - Programnya dapat dilihat di Kode 9.5.

Kode 9.4: Contoh keluaran

```
1 ganjil
2 genap
3 ganjil
4 genap
5 ganjil
6 genap
7 ganjil
8 genap
9 ganjil
10 genap
```

Kode 9.5: Contoh program yang menghasilkan output seperti di Kode 9.4

```
for i in range (10):
    print (i + 1, end = "")
    if (i + 1) % 2 == 0: print(" genap")
    elif (i + 1) % 2 == 1: print (" ganjil")
```

2. Buatlah program yang ketika dijalankan, mencetak 10 baris di terminal (Kode 9.6).
   - Setiap baris dimulai dengan nomor baris (dimulai dari 1).
   - Jika nomor baris kurang dari 6, tertulis karakter "*" sebanyak barisnya.
   - Jika nomor baris lebih dari 5, tertulis karakter "+" sebanyak barisnya.
   - Programnya dapat dilihat di Kode 9.7.

Kode 9.6: Contoh keluaran

```
1 *
2 **
3 ***
4 ****
5 *****
6 ++++++
```

```
7 +++++++
8 ++++++++
9 +++++++++
10 ++++++++++
```

Kode 9.7: Contoh program yang menghasilkan output seperti di Kode 9.6

```
for i in range(10):
    n = i + 1
    print (n, end = ' ')
    karakter = ''
    if n < 6:
        for j in range(n):
            karakter = karakter + '*'
    elif n > 5:
        for j in range(n):
            karakter = karakter + '+'
    print (karakter)
```

3. Buatlah program yang ketika dijalankan, mencetak 20 baris di terminal (Kode 9.8).
   - Setiap baris dimulai dengan nomor baris (dimulai dari 1).
   - Jika nomor baris kelipatan 2, print "kelipatan 2"
   - Jika nomor baris kelipatan 3, print "kelipatan 3"
   - Jika nomor baris kelipatan 2 dan 3, print "kelipatan 2 dan 3"
   - Lainnya, print bintang sebanyak nomor barisnya
   - Programnya dapat dilihat di Kode 9.9.

Kode 9.8: Contoh keluaran

```
1 *
2 kelipatan 2
3 kelipatan 3
4 kelipatan 2
5 *****
6 kelipatan 2 dan 3
7 *******
8 kelipatan 2
9 kelipatan 3
10 kelipatan 2
11 ***********
12 kelipatan 2 dan 3
13 *************
14 kelipatan 2
15 kelipatan 3
16 kelipatan 2
17 *****************
18 kelipatan 2 dan 3
```

```
19 19 ******************
20 20 kelipatan 2
```

Kode 9.9: Contoh program yang menghasilkan output seperti di Kode 9.8

```
for i in range(20):
    n = i + 1
    print (n, end = ' ')
    if n % 2 == 0:
        if n % 3 == 0:
            print ("kelipatan 2 dan 3")
        else:
            print ("kelipatan 2")
    elif n % 3 == 0:
        print ("kelipatan 3")
    else:
        karakter = ''
        for j in range (n):
            karakter = karakter + '*'
        print (karakter)
```

4. Buatlah program yang menerima input berupa angka dari pengguna, lalu program tersebut mencetak baris keluaran sebanyak angka yang dimasukkan. Misal, pengguna memasukkan 12, program mencetak 12 baris dengan ketentuan sebagai berikut.
   - Setiap baris dimulai dengan nomor baris (dimulai dari angka yang dimasukkan pengguna. Jadi, misalnya pengguna memasukkan 12, maka angka di baris pertama adalah 12, lalu 11, lalu 10, dan seterusnya sampai 1).
   - Setelah nomor baris, cetak karakter '*' sebanyak n kali sehingga jumlah nomor baris + $n = 12$. Jadi, di baris pertama tercetak 12 (tanpa bintang), di baris di bawahnya tercetak 11 *, lalu 10 **, dan seterusnya (Kode 9.10).
   - Programnya dapat dilihat di Kode 9.11.

Kode 9.10: Contoh keluaran

```
Masukkan jumlah baris: 12
12
11 *
10 **
9 ***
8 ****
7 *****
6 ******
5 *******
4 ********
```

```
3 *********
2 **********
1 ***********
```

Kode 9.11: Contoh program yang menghasilkan output seperti di Kode 9.10

```
baris = int(input("Masukkan jumlah baris: "))

for i in range (baris, 0, -1):
    n = baris - i
    print (i, end = ' ')
    karakter = ''
    for i in range (n):
        karakter = karakter + '*'
    print (karakter)
```

5. Buatlah program yang menerima input berupa angka dari pengguna, misal 23, lalu program mencetak keluaran 4 angka-angka (ke samping) (Kode 9.12).

Kode 9.12: Contoh keluaran

```
Masukkan angka: 23
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23
```

Kode 9.13: Contoh program yang menghasilkan output seperti di Kode 9.12

```
masukan = int(input("Masukkan angka: "))

counter = 0

for i in range(masukan):
    counter = counter + 1
    print (i + 1, end = ' ')
    if counter == 4:
        counter = 0
        print ('')
print ('')
```