

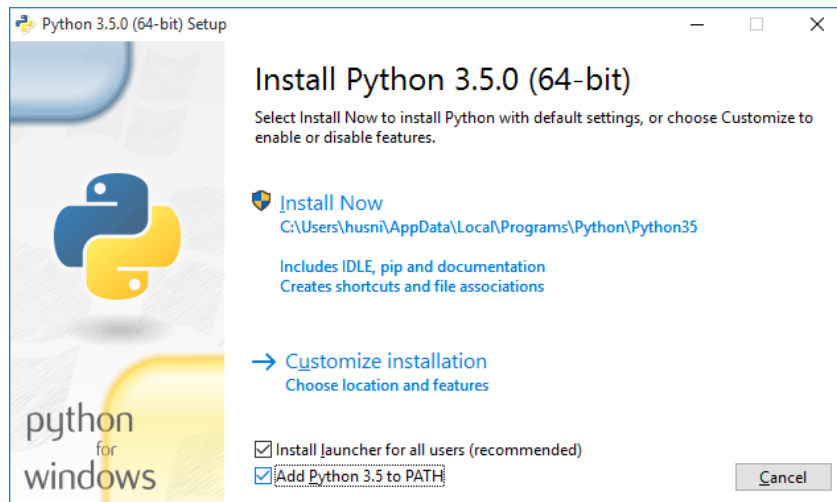
Dasar Pemrograman Python

Husni (husni.trunojoyo.ac.id)

Dasar Pemrograman Python	1
Instalasi Python	2
Memulai Program Python	3
Membuat dan Mengeksekusi Program Python	4
Opsi-opsi Eksekusi Program Python.....	6
Urutan Evaluasi	6
Pernyataan If.....	7
Operator Boolean	8
Hubungan Singkat	9
Pesan Error.....	10
List	11
Komprehensi List.....	12
Pernyataan For	13
Perulangan While.....	14
I Want to Play a Game	15
Praktik Coding	17
Lampiran Lab01.py dan Lab01_extra.py	19

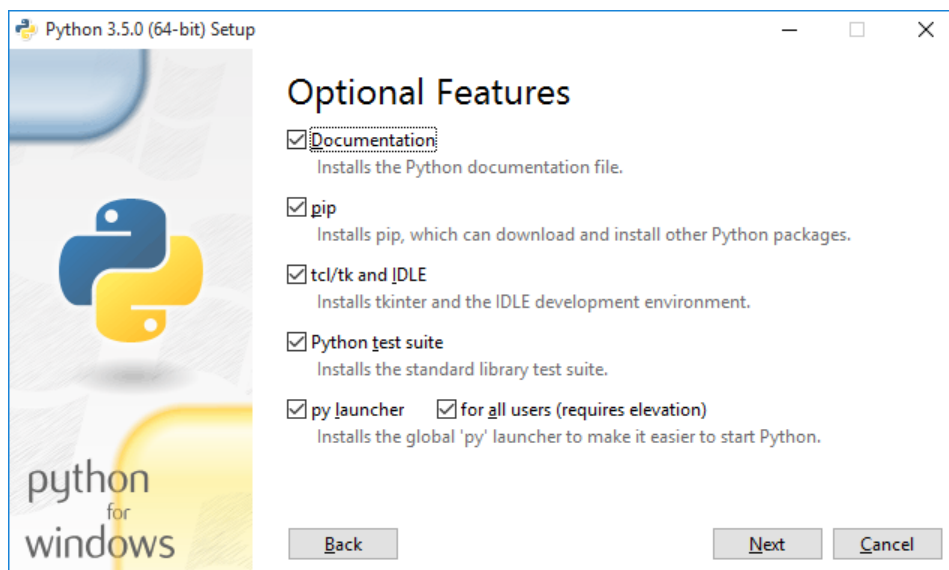
Instalasi Python

Python dapat diinstal dengan mudah. Langkah pertama yang harus dilakukan adalah mendownload python-3.5.0-amd64 atau versi lain yang sesuai dengan kebutuhan dari situsnya di python.org. Ukuran Python terbaru untuk semua versi sekitar 29 MB. Setelah download selesai, cukup *double click* untuk menjalankan program instalasinya.

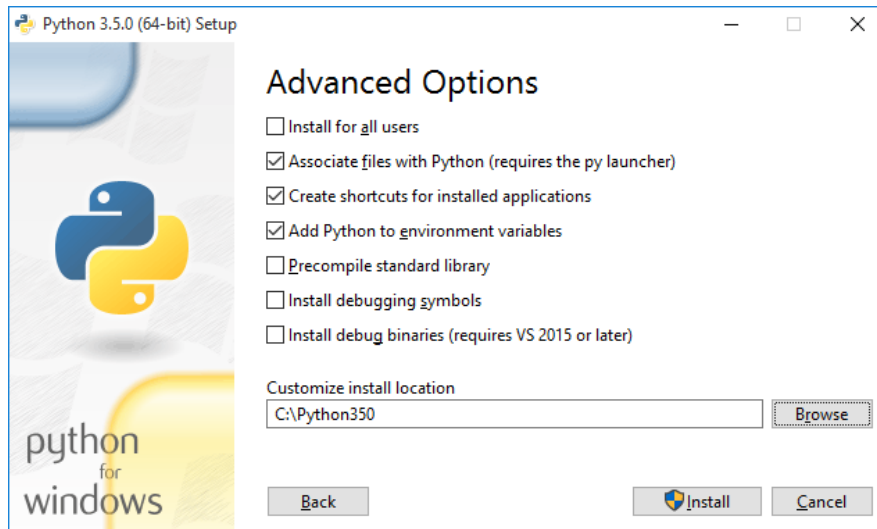


Gambar di atas memperlihatkan jendela pertama dari program instalasi Python. Tidak ada yang sangat perlu dikonfigurasi kecuali satu hal, yaitu memastikan “Add Python 3.5 to PATH” dalam kondisi terpilih (dicentang). Ini maksudnya adalah menambahkan program Python ke dalam PATH dari sistem Windows sehingga dimana pun berada program Python dapat dipanggil untuk mengeksekusi modul-modul yang ditulis mengikuti kaidah bahasa pemrograman Python.

Anda dapat mengarahkan kursor dan klik mouse pada “Install Now” agar instalasi segera dilakukan. Mudah bukan? Namun jika ada yang ingin melakukan beberapa kustomisasi (tidak dianjurkan bagi pemula di Windows) maka boleh memilih “Customize installation”. Ini diperlukan misalnya untuk menginstal Python ke dalam direktori selain C:\Users\Nama_User\AppData\Local\Programs\Python\Python35. Anda akan memperoleh jendela seperti dibawah dan cukup klik Next untuk berpindah ke jendela berikutnya, yaitu jendela Advanced Options.



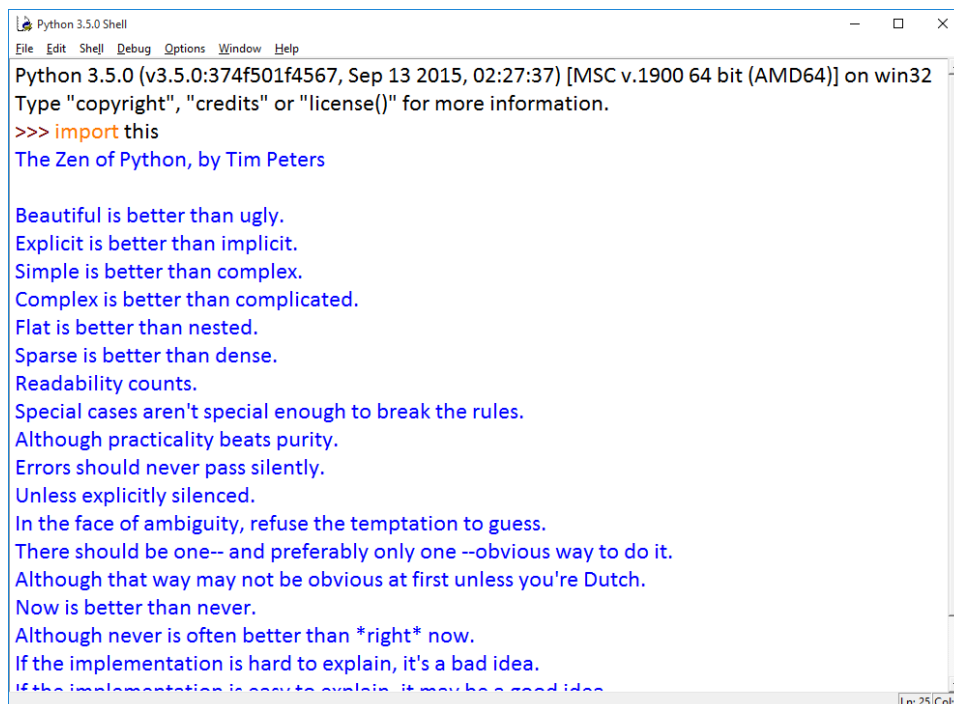
Pada jendela ini, misalnya kita mengubah tujuan instalasi Python ke direktori C:\Python350. Klik Install dan dalam beberapa saat instalasi selesai.



Memulai Program Python

Instalasi Python, selain memberikan kita file-file untuk menjalankan (eksekusi, compile) modul-modul Python, juga menyediakan sebuah program yang sangat penting, yaitu IDLE. IDLE ini dapat digunakan untuk menjalankan perintah Python baris demi baris, selain dapat digunakan untuk menuliskan kemudian mengeksekusi kode program Python yang lengkap dan mungkin terdiri lebih dari 100 baris instruksi.

Silakan jalankan program IDLE sehingga anda sampai di modus Interaktif dari Python. Instruksi yang dikenal oleh Python dapat diberi di sebelah kanan prompt `>>>`. Perhatikan tampilan di bawah ini:



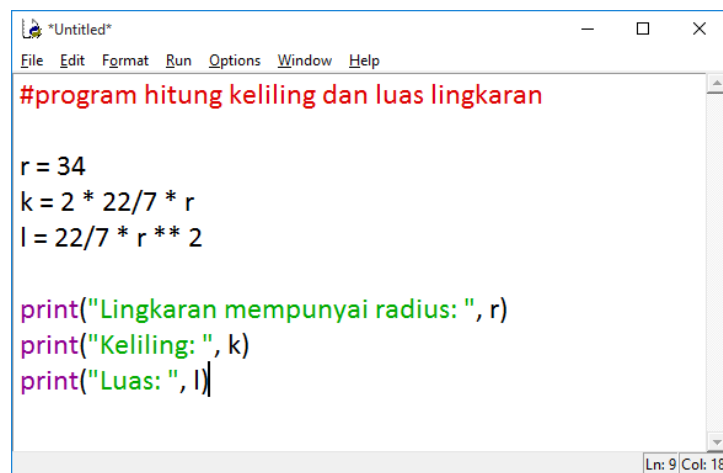
Berikut ini adalah beberapa baris instruksi contoh yang dijalankan di dalam IDLE:

```
>>> nama = 'Husni Ilyas'
>>> nama
>>> print(nama)
>>> umur = 30
>>> print(nama, "berumur ", umur, " tahun.")
```

Apa yang dihasilkan baris-baris di atas? Silakan langsung mencobanya!

Membuat dan Mengeksekusi Program Python

Klik **File | New File** atau tekan **Ctrl N**. Kemudian tuliskan program Python yang diinginkan, misalnya program sederhana berikut untuk menghitung keliling dan luas lingkaran.



```
*Untitled*
File Edit Format Run Options Window Help

#program hitung keliling dan luas lingkaran

r = 34
k = 2 * 22/7 * r
l = 22/7 * r ** 2

print("Lingkaran mempunyai radius: ", r)
print("Keliling: ", k)
print("Luas: ", l)
```

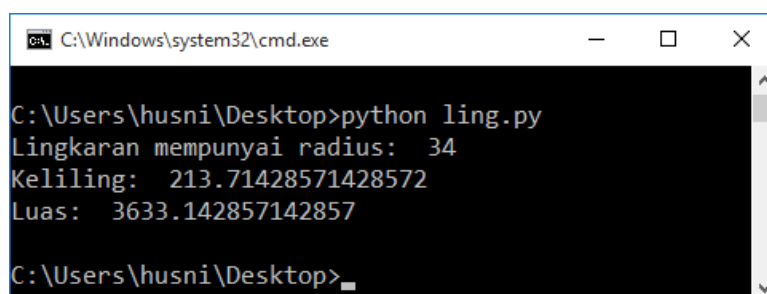
Simpan: Klik **File | Save** atau **Ctrl S**. Beri nama misalnya ling.py dan disimpan di Desktop.

Bagaimana mengeksekusi program atau modul Python tersebut? Klik Menu **Run | Run Module** atau tekan **F5**.

Harusnya diperoleh hasil berikut pada bagian bawah IDLE Editor:

```
===== RESTART: C:\Users\husni\Desktop\ling.py =====
Lingkaran mempunyai radius: 34
Keliling: 213.71428571428572
Luas: 3633.142857142857
```

Apakah modul Python hanya dapat dieksekusi melalui Editor IDLE? TIDAK. Kita dapat menggunakan command line (shell) Windows (begitu pula di Linux, Macintosh dan SO lain). Coba buka Command Prompt Windows, dan pindahkan ke Desktop. Kemudian jalankan perintah python ling.py. Berikut ini adalah capture-nya:

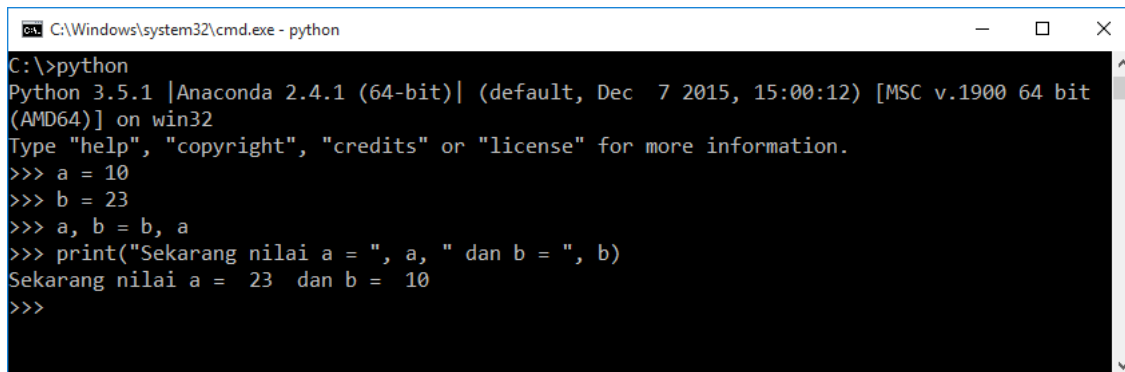


```
C:\Windows\system32\cmd.exe

C:\Users\husni\Desktop>python ling.py
Lingkaran mempunyai radius: 34
Keliling: 213.71428571428572
Luas: 3633.142857142857

C:\Users\husni\Desktop>
```

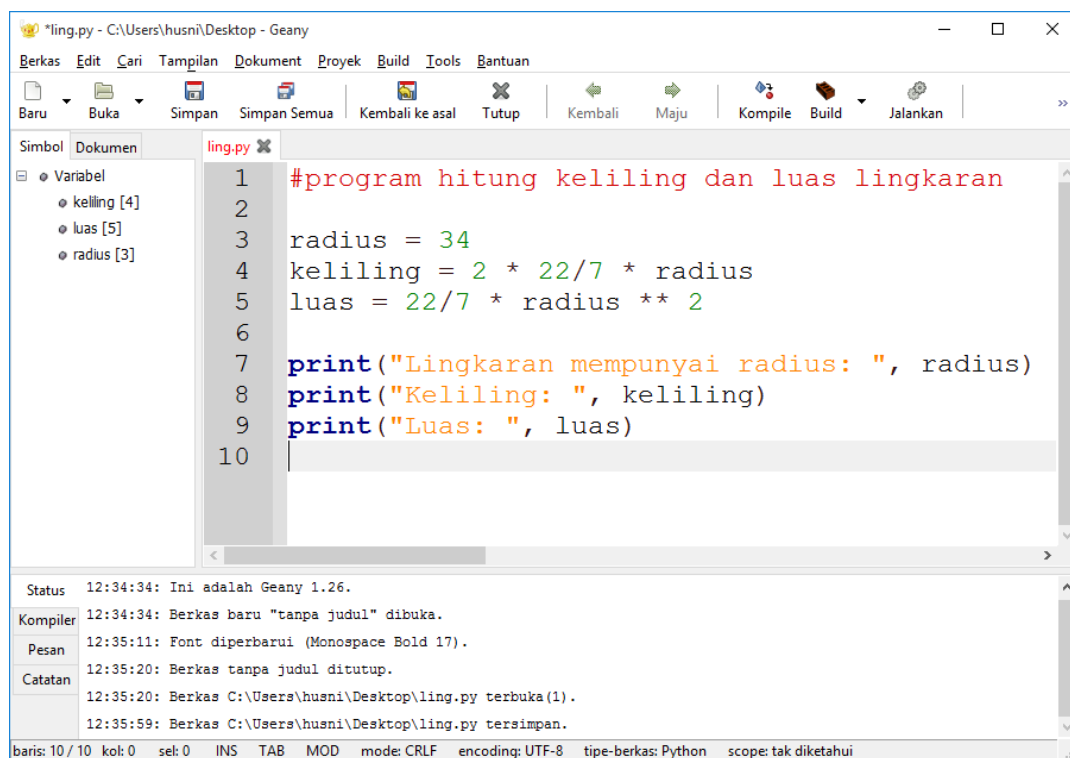
Perintah python selain dapat digunakan untuk mengeksekusi program Python juga dapat dijadikan sebagai shell interaktif sebagaimana di IDLE. Baris-baris di bawah ini memperlihatkan proses menukar data dua variabel di Python (dan hanya berhasil di Python):



```
C:\>python
Python 3.5.1 |Anaconda 2.4.1 (64-bit)| (default, Dec 7 2015, 15:00:12) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 10
>>> b = 23
>>> a, b = b, a
>>> print("Sekarang nilai a = ", a, " dan b = ", b)
Sekarang nilai a = 23 dan b = 10
>>>
```

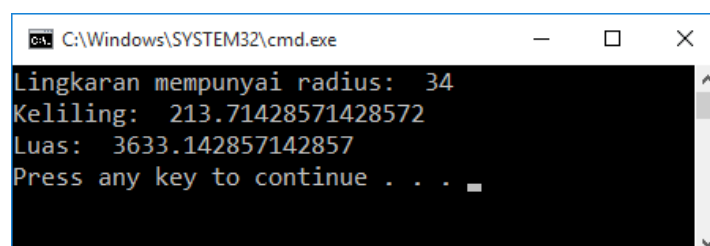
Untuk keluar dari modus Python Interaktif tersebut, gunakan perintah **exit()**.

Apakah penulisan program Python hanya dapat dilakukan di IDLE? TIDAK juga. Kita dapat menggunakan Eclipse, Netbeans, Geany dan Sublime Text yang fenomenal itu. Berikut ini adalah contoh penulisan kode program Python di Geany:



```
*ling.py - C:\Users\husni\Desktop - Geany
Berkas Edit Cari Tampilan Dokument Proyek Build Tools Bantuan
Baru Buka Simpan Simpan Semua Kembali ke asal Tutup Kembali Maju Kompil Build Jalankan
Simbol Dokumen ling.py x
Variabel
  kelling [4]
  luas [5]
  radius [3]
1 #program hitung keliling dan luas lingkaran
2
3 radius = 34
4 keliling = 2 * 22/7 * radius
5 luas = 22/7 * radius ** 2
6
7 print("Lingkaran mempunyai radius: ", radius)
8 print("Keliling: ", keliling)
9 print("Luas: ", luas)
10
Status 12:34:34: Ini adalah Geany 1.26.
Kompiler 12:34:34: Berkas baru "tanpa judul" dibuka.
Pesan 12:35:11: Font diperbarui (Monospace Bold 17).
Catatan 12:35:20: Berkas tanpa judul ditutup.
12:35:20: Berkas C:\Users\husni\Desktop\ling.py terbuka(1).
12:35:59: Berkas C:\Users\husni\Desktop\ling.py tersimpan.
baris: 10 / 10 kol: 0 sel: 0 INS TAB MOD mode: CRLF encoding: UTF-8 tipe-berkas: Python scope: tak diketahui
```

Bagaimana eksekusinya? Cukup klik **Build | Execute** atau tekan **F5**. Hasilnya adalah eksekusi perintah Python dari Command Line Windows:



```
C:\Windows\SYSTEM32\cmd.exe
Lingkaran mempunyai radius: 34
Keliling: 213.71428571428572
Luas: 3633.142857142857
Press any key to continue . . .
```

Opsi-opsi Eksekusi Program Python

Pada saat menjalankan atau mengeksekusi file Python (berakhiran .py), kita dapat menggunakan **flags** pada Command Line untuk menginspeksi kode program lebih lanjut. Bagian ini akan memperlihatkan sedikit flag yang penting, sisanya silakan membaca dokumentasi yang sudah ikut terinstall bersama Python.

- Pemanggilan file Python tanpa flag akan menjalankan kode yang ada di dalam file dan mengembalikan kita ke Command Line (seperti diperlihatkan sebelumnya).

```
python3 ling.py          # pada sistem yang terdapat dua Python (versi 2 dan 3)
python ling.py
```

- Opsi atau flag `-i` menjalankan script Python, kemudian membuka suatu sesi interaktif. Untuk keluar silakan ketik `exit()` pada prompt interpreter yang tersedia. Kita dapat juga menggunakan shortcut keyboard `Ctrl-D` pada mesin Linux/Mac atau `Ctrl-Z` Enter pada sistem Windows. Jika anda mengedit file python saat sedang dijalankan secara interaktif, anda perlu keluar dan merestart interpreter agar perubahan tersebut memberikan pengaruhnya.

```
python3 -i ling.py
```

- Opsi `-m doctest` akan menjalankan doctests di dalam file tertentu. Doctests ditandai dengan *triple quotes* (`"""`) yang biasanya diletakkan di dalam fungsi atau kelas.

```
python3 -m doctest ling.py
```

Urutan Evaluasi

Dalam urutan mengevaluasi suatu ekspresi, pertama adalah evaluasi terhadap ekspresi *operand* untuk menghasilkan suatu nilai, kemudian mengevaluasi operator pada nilai-nilai *operand*. Jika melibatkan fungsi, evaluasi terhadap *operand* dilakukan sebelum evaluasi terhadap fungsi yang melibatkan nilai-nilai yang dihasilkan. Definisi rekursif secara natural ini adalah salah satu konsep paling esensi dalam komputasi. Ini mengharuskan kita berpikir mengenai program tersebut dari sudut pandang komputer. Ini diluar dari *syntax* (bagaimana itu dituliskan) dan *semantics* (apa makna dari yang dituliskan).

Kita sudah melihat pernyataan *assignment* (menggunakan tanda =) yang digunakan untuk memberikan suatu nilai (atau hasil dari suatu ekspresi) ke suatu variabel. Tetapi pernyataan *assignment* itu sendiri tidak mempunyai nilai. Ini bermanfaat dikarenakan adanya kontrol *sequential*. Nilai yang diberikan ke variabel tersebut dapat digunakan dalam ekspresi dan pernyataan (*statement*) nantinya dengan merujuk variabel tersebut. Berikut ini adalah contoh sebuah fungsi bernama **quadratic** yang melibatkan 4 variabel: a, b, c dan t dan satu fungsi **sqrt** untuk mendapatkan akar dari parameternya:

```
def quadratic(a, b, c):
    t = sqrt(b * b - 4 * a * c)
    return (-b - t) / (2 * a), (-b + t) / (2 * a)
```

Pernyataan If

Conditional statement adalah suatu pernyataan, bukan suatu ekspresi; pernyataan tidak mengembalikan suatu nilai. Ekspresi if (atau predikat) dievaluasi terlebih dahulu, sebelum bagian lain dari pernyataan, untuk menentukan apakah mengevaluasi suatu lengan. Jika ekspresi if terevaluasi bernilai True maka pernyataan yang mengikuti: dievaluasi. Jika tidak, lengan else yang dievaluasi, jika ada. Banyak predikat dapat disambung bersama dengan elif. Semua predikat itu diavaluasi secara sequential. Pernyataan kondisional sering digunakan bersama dengan pernyataan return di dalam fungsi. Sebagai contoh, dalam beberapa data sensus terlihat decode gender berikut:

```
def decode_gender(code):
    if (code == 0):
        return 'all'
    elif (code == 1):
        return 'male'
    elif (code == 2):
        return 'female'
    else:
        return 'unknown'
```

Kondisional sering digunakan dengan pernyataan assignment untuk menyederhanakan ekspresi:

```
if ((year % 4) == 0) and (((year % 100) != 0) or ((year % 400) == 0)):
    year_len = 366
else:
    year_len = 365
<lakukan sesuatu dengan year_len>
```

Atau dengan pernyataan print untuk mengontrol output:

```
if (scene == 'architect skit'):
    print("spam, spam, spam")
else
    print("nobody expects the Spanish inquisition")
```

Pernyataan if sering tidak tepat digunakan oleh pemula untuk menyusun ekspresi boolean sederhana.

Hint: Kadang kita melihat kode seperti berikut:

```
if x > 3:
    return True
else:
    return False
```

Ini dapat ditulis lebih tepat sebagai `return x > 3`.

Fungsi berikut didefinisikan di dalam file lab01.py (lihat file lab01.py di akhir tutorial ini). Jika sulit memahaminya, coba gunakan `python3 -i lab01.py`.

```
>>> def xk(c, d):
...     if c == 4:
...         return 6
...     elif d >= 4:
```

```

...     return 6 + 7 + c
...     else:
...         return 25
>>> xk(10, 10)
>>> xk(10, 6)
>>> xk(4, 6)
>>> xk(0, 0)

>>> def how_big(x):
...     if x > 10:
...         print('huge')
...     elif x > 5:
...         return 'big'
...     elif x > 0:
...         print('small')
...     else:
...         print("nothin'")
>>> how_big(7)
>>> how_big(12)
>>> how_big(1)
>>> how_big(-1)

>>> def so_big(x):
...     if x > 10:
...         print('huge')
...     if x > 5:
...         return 'big'
...     if x > 0:
...         print('small')
...     print("nothin'")
>>> so_big(7)
>>> so_big(12)
>>> so_big(1)

```

Hint: `print` (tidak seperti `return`) tidak menyebabkan keluar dari fungsi!

Operator Boolean

Kita telah melihat pemanfaatan operator boolean di atas, di bagian ini kita akan melihat koneksi antara beberapa operator tersebut dan kondisional.

Python mendukung 3 operator boolean: `and`, `or` dan `not`. Coba jalankan instruksi-instruksi berikut menggunakan IDLE (modus interaktif):

```

>>> a = 4
>>> a < 2 and a > 0
>>> a < 2 or a > 0
>>> not (a > 0)

```

`and` dievaluasi `True` hanya jika kedua operand terevaluasi `True`. Jika setidaknya satu operand bernilai `False`, maka `and` terevaluasi sebagai `False`.

or dievaluasi menjadi True jika setidaknya satu operand terevaluasi sebagai True. Jika semua *operand* bernilai False, maka or terevaluasi False.

Bagaimana pendapat anda terhadap evaluasi ekspresi berikut? Coba buktikan dengan Python.

```
>>> True and not False or not True and False
```

Adalah sulit membaca ekspresi yang kompleks, seperti di atas, dan memahami bagaimana suatu program akan berperilaku. Penggunaan tanda kurung dapat membuat kode lebih mudah dipahami. Python menginterpretasi ekspresi di atas dengan cara berikut:

```
>>> (True and (not False)) or ((not True) and False)
```

Ini adalah karena operator boolean, seperti operator aritmatika, mempunyai suatu urutan operasi:

- not mempunyai prioritas tertinggi
- and
- or mempunyai prioritas paling rendah

Untuk membuat kode dapat dibaca (*readable*), and dan or bekerja lebih banyak daripada boolean saja (True, False). Nilai Python lain dapat dipertimbangkan "f-false" termasuk 0, None dan " (string kosong). Semua nilai selain itu dianggap "y-true."

```
>>> 0 or True
```

```
>>> not " or not 0 and False
```

```
>>> 13 and False
```

Hubungan Singkat

Menurut anda, apa yang akan terjadi jika kita menulis baris berikut di dalam IDLE (Python)?

```
1 / 0
```

Sebagaimana diduga, kita memperoleh error pembagian oleh nol, tepatnya `ZeroDivisionError`. Tetapi bagaimana dengan ekspresi berikut?

```
True or 1 / 0
```

Itu terevaluasi True karena operator and dan or pada Python bersifat *short-circuit* (hubungan singkat). Maksudnya, tidak perlu dilakukan evaluasi terhadap setiap *operand*.

Operator	Evaluasi dari kiri ke kanan sampai:	Contoh
AND	Nilai "y-false" pertama	False and 1 / 0 terevaluasi False
OR	Nilai "y-true" pertama	True or 1 / 0 terevaluasi True

Jika and dan or tidak *short-circuit* maka mengembalikan nilai terakhir. Ini berarti bahwa and dan or tidak selalu mengembalikan booleans ketika menggunakan nilai y-true dan y-false.

Berikut ini adalah contoh kecilnya:

```
def divides(x, y):  
    """  
    kembalikan True jika x membagi y  
    """
```

```
return x and y % x == 0
```

Hint: Jika terjadi error, catat Error tersebut.

```
>>> True and 13
>>> False or 0
>>> not 10
>>> not None

>>> True and 1 / 0 and False
>>> True or 1 / 0 or False
>>> True and 0
>>> False or 1
>>> 1 and 3 and 6 and 10 and 15
>>> 0 or False or 2 or 1 / 0
```

Pertanyaan 1: Membetulkan Bug

Potongan kode berikut tidak bekerja dengan baik! Jelaskan apa salahnya dan perbaiki bug-nya.

```
def both_positive(x, y):
    """Kembalikan True jika x dan y bernilai positif.

    >>> both_positive(-1, 1)
    False
    >>> both_positive(1, 1)
    True
    """
    return x and y > 0 # Anda dapat mengganti baris ini!
```

Jawaban 1:

```
return x > 0 and y > 0
```

Baris aslinya (`return x and y > 0`) akan memeriksa apakah dua hal berikut bernilai true:

```
x
```

```
y > 0
```

Kapan `x` akan dianggap True? Dalam Python, suatu bilangan yang tidak 0 akan dianggap True. Jadi, doctest pertama akan gagal: `x = -1` dan `-1 != 0`, dan `y = 1 > 0`, sehingga keduanya menyebabkan True.

Pesan Error

Sejauh ini kita sudah melihat dua pesan kesalahan (error). Pesan error terasa mengganggu tetapi sebetulnya sangat bermanfaat dalam code debugging. Berikut ini adalah beberapa jenis error yang umum terjadi saat eksekusi kode Python:

Jenis Error	Penjelasan
SyntaxError	Terdapat sintaks yang tidak tepat (misal hilangnya titik dua setelah pernyataan if atau lupa menutup kurung atau quote)
IndentationError	Terdapat kesalahan indentasi (misal indentasi tak konsisten dalam suatu badan fungsi)

TypeError	Operasi yang dicoba pada tipe data tak-kompatibel (misal mencoba untuk menambahkan fungsi dengan suatu bilangan) atau memanggil fungsi dengan jumlah argumen yang salah
ZeroDivisionError	Percobaan pembagian dengan nol

Menggunakan deskripsi pesan-pesan error ini, kita akan mendapatkan ide yang lebih baik mengenai error apa yang akan terjadi dengan kode program. **Jika anda mendapatkan pesan error, cobalah untuk mengidentifikasi masalah sebelum meminta bantuan orang lain.** Anda boleh jadi akan sering memanfaatkan Google dan menemukan pesan error yang tak umum yang telah terjadi pada banyak orang sebelumnya dan bagaima mereka men-debug-nya.

Sebagai contoh:

```
>>> square(3, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: square() takes 1 positional argument but 2 were given
```

Catatan:

Baris terakhir dari pesan error memberitahukan kita jenis dari error. Dalam contoh di atas, diketahui jenis errornya `TypeError`. Pesan error memberitahukan kita apa kesalahan yang telah kita lakukan, yaitu memberikan fungsi `square` 2 argumen padahal ia hanya dapat menerima 1 argumen. Secara umum, baris terakhir sangat bermanfaat.

Baris kedua terakhir dari pesan error memberitahukan kita baris dimana error terjadi. Ini membantu kita menjelajah error tersebut. Pada contoh di atas, `TypeError` terjadi pada baris 1.

List

Sejauh ini, kita telah bermain-main dengan tipe data skalar (`int`, `float` dan `boolean`) dan satu tipe deretan (`string`) yang banyak bertingkah seperti suatu skalar. List merupakan tipe deretan (`sequence`) yang sangat fleksibel. **Suatu list mengandung sederetan nilai bertipe tertentu.**

Kita dapat menentukan list hanya dengan menempatkan nilai-nilai yang dipisahkan koma di dalam suatu kurung siku. Data sering hadir dalam beberapa jenis di dalam list. Berikut ini contohnya (silakan langsung dicoba secara interaktif):

```
>>> [1,2,3]
>>> ["frog", 3, 3.1415]
>>> [True, [1, 2], 42]
```

Sekarang, tetapkan di IDLE dan masukkan instruksi-instruksi berikut dan perhatikan apa yang diberikan oleh Python:

```
>>> x = [1,2,3]          # memberikan list ke suatu variabel
>>> len(x)              # mengambil panjangnya, yaitu jumlah elemen di dalam list
>>> x + [4,5]           # + adalah penggabungan
>>> [1,2]*3             # * adalah replikasi (penggandaan)
>>> len([1,2]*3)
>>> [1,2]*[3,4]         # Apa yang dilakukan dan apa hasilnya?
TypeError: can't multiply sequence by non-int of type 'list'
```

Operator `in` bersifat sangat *cool*. Ia beroperasi pada list lengkap dan menghasilkan suatu boolean yang menjawab pertanyaan "Adakah item ini di dalam list?". Coba langsung baris-baris instruksi berikut:

```
>>> 2 in [1,2,3]
>>> "frog" in [1,2,3]
>>> [1,2] in [1,2,3]
>>> [1,2] in [[1,2],3]
```

Komprehensi List

Kita sudah dapat membuat list, memberikannya ke suatu variabel, menuliskan ekspresi dan mendefinisikan fungsi. Kita dapat menyatukan konsep-konsep ini untuk melakukan banyak hal menarik. Komprehensi list di Python membuka suatu keindahan pemrograman berpusat pada data (*data-centric programming*).

Komprehensi ada di dalam kurung siku, seperti list, tetapi tidak berupa deretan literal statis (ia adalah list yang terkomputasi secara dinamis). Apa yang dihasilkan 3 baris ini? Cobalah!.

```
>>> somelist = [1, 2, 9, -1, 0]           # ini list
>>> [x+1 for x in somelist]             # ini komprehensi
>>> [x*x for x in somelist]             # ini komprehensi
```

Secara umum, ekspresi di dalam `[` dievaluasi setiap elemennya menggunakan variabel antara `for` dan `in` untuk menamai setiap elemen. Hasilnya adalah list yang bertransformasi.

```
>>> def square(x):
...     return x*x
...
>>> def squares(s):
...     return [square(x) for x in s]
...
>>> squares([0,1,2,4])
[0, 1, 4, 16]
```

Inilah suatu pola rancangan (*design pattern*) yang tangguh, dinamakan `map`, yang akan sering digunakan dalam menganalisa data. Ia memetakan atau mengtransformasi satu struktur data ke dalam bentuk lain dengan beberapa ekspresi, seringnya dengan menerapkan fungsi untuk setiap elemen.

Kadang kala kita memerlukan suatu deretan untuk memulainya, dan Python menyediakan tool untuk itu. Salah satunya adalah `range`.

```
>>> [x*x for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Pertanyaan 2: Mengelompokkan Elemen-elemen

Lengkapilah fungsi `odd_even` yang mengelompokkan suatu bilangan sebagai 'odd' (ganjil) atau 'even' (genap) dan fungsi `classify` yang mengambil elemen-elemen dalam list dan menerapkan `odd_even` terhadap semua elemen dari list tersebut.

```
def odd_even(x):
```

```

"""
Kelompokkan suatu bilangan sebagai odd (ganjil) atau even (genap).
>>> odd_even(4)
'even'
>>> odd_even(3)
'odd'
"""
"""*** KODE ANDA DI SINI ***"""

```

```

def classify(s):
    """
    Kelompokkan semua elemen dari suatu sequence sebagai odd atau even
    >>> classify([0, 1, 2, 4])
    ['even', 'odd', 'even', 'even']
    """
    """*** KODE ANDA DI SINI ***"""

```

Jawaban 2:

```

def odd_even(x):
    """
    Kelompokkan suatu bilangan sebagai odd atau even.
    >>> odd_even(4)
    'even'
    >>> odd_even(3)
    'odd'
    """
    if (x % 2) == 0:
        return 'even'
    else:
        return 'odd'

def classify(s):
    """
    Kelompokkan semua elemen dari sequence sebagai odd atau even
    >>> classify([0, 1, 2, 4])
    ['even', 'odd', 'even', 'even']
    """
    return [odd_even(x) for x in s]

```

Pernyataan For

Python menawarkan suatu pernyataan `for` yang sangat mirip dengan `for` di dalam komprehensi list, tetapi mengontrol urutan evaluasi sesuatu yang disebut iterasi. Pada dasarnya, ini digunakan untuk melakukan iterasi sepanjang suatu list selama komputasi sesuatu. Berikut ini adalah contohnya:

```

def sum(s):
    """
    Mengembalikan jumlah dari elemen dalam deretan s
    """
    psum = 0          # status awal: jumlah elemen dalam deretan null
    for x in s:      # untuk setiap elemen dalam deretan atau sequence

```

```
psum = psum + x    # akumulasikan ia ke dalam partial sum
return psum        # partial sum final adalah jumlah total
```

Tentu saja, pernyataan akumulasi sudah sangat umum mempunyai shorthand berikut:

```
psum += x
```

Di satu sisi, `for` di dalam komprehensi list memperkenalkan pola rancangan *map* (pemetaan), pernyataan `for` memperkenalkan pola rancangan esensi kedua untuk analisa data, *reduce* (pengurangan). Anda akan menemukan bahwa *map-reduce* adalah struktur komputasional yang penting sekali dalam ilmu pengetahuan mengenai data (*data science*).

Pertanyaan 3: Penjumlahan Akar

Lengkapi fungsi di bawah ini untuk menghitung jumlah akar.

```
def sum_of_squares(n):
    """
    Mengembalikan jumlah dari akar dari 1 sampai n
    >>> sum_of_squares(3)
    14
    """
    """ KODE ANDA DI SINI """
```

Jawaban 3:

```
psum = 0
for x in range(1,n+1):
    psum += x*x
return psum
```

Perulangan While

Python juga mempunyai suatu konstruksi iterasi dasar lain yang berkaitan erat dengan kondisional, perulangan (loop) *while*. Tidak ada asumsi iterasi terhadap *sequence*. Iterasi dilakukan sampai suatu predikat terpenuhi, bukan sebanyak jumlah elemen dalam deretan.

Pada dasarnya, beberapa status akan dibangun sebelum perulangan *while*. Predikat akan menghitung suatu ekspresi boolean yang menyertakan status tersebut. Dan badan dari perulangan akan menaikkan status tersebut, dengan demikian pengiterasian sampau predikat terpenuhi.

Pertanyaan 4: Perulangan

```
>>> n = 3
>>> while n >= 0:
...     n -= 1
...     print(n)
```

Hint: Pastikan kondisi perulangan *while* suatu waktu akan bernilai *false*, jika tidak maka perulangan tidak akan pernah berhenti. Menekan `Ctrl C` akan menghentikan perulangan tanpa batas secara paksa oleh interpreter.

```
>>> n = 4
>>> while n > 0:
...     n += 1
```

```

... print(n)
>>> def go(n):
...     if n % 2 != 0:
...         print(n / 2)
...         return
...     while n > 0:
...         print(n)
...         n = n // 2
>>> go(4)
>>> go(5)

```

Coba dipikirkan: apakah mungkin membuat suatu program untuk mendeteksi adanya *loop* (perulangan) tanpa batas di dalam suatu program? Atau dapatkah anda membuat suatu program yang akan menghentikan program lain?

I Want to Play a Game

Algoritma adalah sehimpunan langkah-langkah untuk menyelesaikan suatu tugas. Kita menggunakan algoritma setiap hari, seperti menambahkan bilangan dengan tangan untuk mendapatkan nama haripada 5 hari dari sekarang.

Mari kita mencoba bermain tebak-an angka dengan Python. Ambil satu bilangan dan Python akan menebak secara random sampau tebakannya benar (bukankita yang menebaknya).

Semua kode untuk game ini ada di dalam `lab01.py`. Pada Command Line, mulailah suatu sesi interaktif dari Python (refreshing, tidak menggunakan IDLE, meskipun hasilnya sama saja) dan jalankan perintah berikut:

```
$ python3 -i lab01.py
```

Fungsi `guess_random` akan meminta kita (pengguna) memberikan suatu bilangan, menanyakan apakah tebak-an dari komputer sudah benar (banyak kali) dan mengembalikan bilangan yang ditebak oleh Python. Untuk memberitahukan Python apakah tebakannya sudah benar, cukup masukkan `y` pada prompt `[y/n]`. Jika tebak-an masih salah, masukkan `n`. Python belum mampu menebak dengan baik, sehingga eksekusinya memakan waktu lama. Tekan `Ctrl C` untuk memaksakan program ini selesai (dihentikan).

```

>>> guess_random()
Pick an integer between 1 and 10 (inclusive) for me to guess: 7
Is 1 your number? [y/n] n
Is 5 your number? [y/n] n
...

```

Meskipun tebak-an random ini bekerja baik, kita dapat membuat strategi tebak-an yang lebih baik.

Question 5: Tebak-an Linier

Satu kelemahan dari strategi `guess_random` adalah terjadinya pengulangan tebak-an yang salah. Daripada tebak-an secara liar, coba buat angka tebak-an urut naik.

Catatan: `is_correct` adalah fungsi yang akan mengembalikan `True` jika tebak-an cocok dengan bilangan yang benar. Silakan merujuk implementasi `guess_random` untuk menerapkan `guess_linear`.

```
def guess_linear():
```

```

"""Tebakan dalam urutan naik dan mengembalikan bilangan yang ditebak."""
prompt_for_number(LOWER, UPPER)
num_guesses = 1
guess = LOWER
"""*** KODE ANDA DI SINI ***"""
return num_guesses

```

Jawaban 5:

```

while not is_correct(guess):
    guess += 1
    num_guesses += 1

```

Cara terbaik untuk menguji fungsi ini adalah dengan memainkan game secara interaktif. Apakah algoritma sudah bekerja sesuai yang diharapkan?

Pertanyaan 6: Tebakan Biner

Pertanyaan Menantang. Fungsi `guess_linear` dapat menghabiskan waktu yang lama jika bilangan yang akan ditebak sangat besar. Namun, suatu strategi dinamakan *binary search* dapat menemukan bilangan dengan lebih cepat. Idennya adalah memulai di tengah range dan setelah setiap tebakan yang tidak benar menanyakan apakah tebakannya terlalu tinggi (`is_too_high`) atau terlalu rendah. Salah satu cara, kita dapat mengeliminasi setengah sisa tebakan yang mungkin.

Hint: Coba gunakan fungsi `is_too_high` untuk menerapkan strategi yang lebih cepat.

`is_too_high` akan mengembalikan `True` jika tebakan lebih besar daripada bilangan yang benar. Seperti contoh di bawah ini:

```

>>> result = is_too_high(5)
Is 5 too high? [y/n] y
>>> result
True

```

Hint: Anda mungkin ingin mengupdate variabel lain diluar `guess`.

```

def guess_binary():
    """Return the number of attempted guesses. Implement a faster search
    algorithm that asks the user whether a guess is less than or greater than
    the correct number.

    Hint: If you know the guess is greater than the correct number, then your
    algorithm doesn't need to try numbers that are greater than guess.
    """
    prompt_for_number(LOWER, UPPER)
    num_guesses = 1
    lower, upper = LOWER, UPPER
    guess = (lower + upper) // 2
    """*** KODE ANDA DI SINI ***"""
    return num_guesses

```

Jawaban 6:

```

while not is_correct(guess):
    if is_too_high(guess):

```



```
upper = guess - 1
else:
    lower = guess + 1
    guess = (lower + upper) // 2
    num_guesses += 1
```

Jika kita memilih suatu bilangan antara 1 dan 10, pendekatan ini tidak perlu lebih dari 4 tebakan untuk menemukan bilangan tersebut.

Cara terbaik menguji fungsi ini adalah dengan memainkannya secara interaktif. Cobalah berpikir *edge cases* — bilangan-bilangan yang mungkin menyebabkan algoritma tersebut melakukan kesalahan. Jika anda mengedit file Python saat menjalankannya secara interaktif maka perlu untuk `exit()` dan me-restart interpreter agar perubahan yang dilakukan mempengaruhi eksekusi.

Sejauh ini, algoritma di atas hanya harus menemukan bilangan antara 1 dan 10. Bagaimana jika kita perluas cakupannya antara 1 dan 100? Berapa banyak tebakan yang dibuat oleh setiap algoritma jika 100 diambil sebagai bilangan terpilih?

Praktik Coding

Kode untuk bagian ini dapat ditemukan di dalam file `lab01_extra.py`.

Sebaiknya praktik ini dapat diselesaikan di lab atau di rumah dalam waktu 2 jam. Jika tidak tuntas, sangat diharapkan anda menyelesaikannya dengan coding bersama teman memanfaatkan waktu senggang yang tersedia.

Pertanyaan 7: Faktorial

Implementasikan suatu fungsi `factors` yang meminta bilangan `n` sebagai parameter dan mencetak (dalam urutan menurun) semua bilangan yang habis membagi `n` secara genap. Sebagai contoh, faktorial dari 20 adalah 20, 10, 5, 4, 2, 1.

```
def factors(n):
    """Prints out all of the numbers that divide `n` evenly.

    >>> factors(20)
    20
    10
    5
    4
    2
    1
    """
    """*** KODE ANDA DI SINI ***"""
```

Jawaban 7:

```
x = n
while x > 0:
    if n % x == 0:
        print(x)
    x -= 1
```

Pertanyaan 8: Faktorial Menurun

Tulislah suatu fungsi bernama `falling` untuk mendapatkan faktorial "menurun" dari suatu bilangan. Fungsi ini memerlukan dua argumen `n` dan `k` dan mengembalikan hasil perkalian sebanyak `k` kali dimulai dari bilangan `n` dan bekerja secara menurun.

```
def falling(n, k):
    """Compute the falling factorial of n to depth k.

    >>> falling(6, 3) # 6 * 5 * 4
    120
    >>> falling(4, 0)
    1
    >>> falling(4, 3) # 4 * 3 * 2
    24
    >>> falling(4, 1) # 4
    4
    """
    """*** KODE ANDA DI SINI ***"""
```

Jawaban 8:

```
total, stop = 1, n-k
while n > stop:
    total, n = total*n, n-1
return total
```

Lampiran Lab01.py dan Lab01_extra.py

Lab01.py:

```
"""Lab 1."""

# If Statements

def xk(c, d):
    if c == 4:
        return 6
    elif d >= 4:
        return 6 + 7 + c
    else:
        return 25

def how_big(x):
    if x > 10:
        print('huge')
    elif x > 5:
        return 'big'
    elif x > 0:
        print('small')
    else:
        print("nothin")

def so_big(x):
    if x > 10:
        print('huge')
    if x > 5:
        return 'big'
    if x > 0:
        print('small')
    print("nothin")

# Boolean Operators

def both_positive(x, y):
    """Returns True if both x and y are positive.

    >>> both_positive(-1, 1)
    False
    >>> both_positive(1, 1)
    True
    """
    return x and y > 0 # You can replace this line!
```

```

# Classify elements

def odd_even(x):
    """
    Classify a number as odd or even.
    >>> odd_even(4)
    'even'
    >>> odd_even(3)
    'odd'
    """
    """
    *** KODE ANDA DI SINI ***
    """

def classify(s):
    """
    Classify all the elements of a sequence as odd or even
    >>> classify([0, 1, 2, 4])
    ['even', 'odd', 'even', 'even']
    """
    """
    *** KODE ANDA DI SINI ***
    """

# For Statements

def sum_of_squares(n):
    """
    return the sum of squares from 1 to n
    >>> sum_of_squares(3)
    14
    """
    """
    *** KODE ANDA DI SINI ***
    """

# Guessing Games

from random import randint

LOWER = 1
UPPER = 10

def guess_random():
    """Guess randomly and return the number of guesses."""
    prompt_for_number(LOWER, UPPER) # asks the user to choose a number
    num_guesses, correct = 0, False
    while not correct:
        guess = randint(LOWER, UPPER) # randomly guess
        correct = is_correct(guess) # ask user if guess is correct
        num_guesses = num_guesses + 1
    return num_guesses

def guess_linear():
    """Guess in increasing order and return the number of guesses."""
    prompt_for_number(LOWER, UPPER)
    num_guesses = 1

```

```

guess = LOWER
*** KODE ANDA DI SINI ***
return num_guesses

def guess_binary():
    """Return the number of attempted guesses. Implement a faster search
    algorithm that asks the user whether a guess is less than or greater than
    the correct number.

    Hint: If you know the guess is greater than the correct number, then your
    algorithm doesn't need to try numbers that are greater than guess.
    """

    prompt_for_number(LOWER, UPPER)
    num_guesses = 1
    lower, upper = LOWER, UPPER
    guess = (lower + upper) // 2
    *** KODE ANDA DI SINI ***
    return num_guesses

# Receive user input. You do not need to understand the code below this line.

def prompt_for_number(lower, upper):
    """Prompt the user for a number between lower and upper. Return None."""
    is_valid_number = False
    while not is_valid_number:
        # You don't need to understand the following two lines.
        number = input('Pick an integer between {0} and {1} (inclusive) for me to guess: '.format(lower,
upper))
        number = int(number)
        if lower <= number <= upper:
            is_valid_number = True

def is_correct(guess):
    """Ask the user if a guess is correct and return whether they respond y."""
    return is_yes('Is {0} your number? [y/n] '.format(guess))

def is_too_high(guess):
    """Ask the user if a guess is too high and return whether they say yes."""
    return is_yes('Is {0} too high? [y/n] '.format(guess))

def is_yes(prompt):
    """Ask the user a yes or no question and return whether they say yes."""
    while True: # This while statement will loop until a "return" is reached.
        yes_no = input(prompt).strip()
        if yes_no == 'y':
            return True
        elif yes_no == 'n':
            return False
    print('Please type y or n and press return/enter')

```

Lab01_extra.py:

```
"""Coding practice for Lab 1."""

# While Loops

def factors(n):
    """Prints out all of the numbers that divide `n` evenly.

    >>> factors(20)
    20
    10
    5
    4
    2
    1
    """
    """*** KODE ANDA DI SINI ***"""

def falling(n, k):
    """Compute the falling factorial of n to depth k.

    >>> falling(6, 3) # 6 * 5 * 4
    120
    >>> falling(4, 0)
    1
    >>> falling(4, 3) # 4 * 3 * 2
    24
    >>> falling(4, 1) # 4
    4
    """
    """*** KODE ANDA DI SINI ***"""
```