

BELAJAR PEMROGRAMAN DALAM BAHASA PYTHON

Tulisan ini dibuat untuk memperkenalkan Anda kepada pemrograman dengan Python, dengan banyak praktik 'coding' serta mengajak Anda untuk belajar lebih dalam secara mandiri dengan contoh yang lebih rumit daripada yang diperlukan pada topik yang dibahas.

Referensi utama dari buku Dr. Charles R. Severance berjudul Python for Everybody atau publikasinya py4e di github. Referensi lain adalah Think Python oleh Allen B. Downey; Algorithmic Problem Solving with Python oleh Schneider, Broschat, Dahmen

Mengapa belajar pemrograman dengan Python

Komputer telah menjadi hal yang lazim saat ini, seperti obeng dan tang, hanya lebih rumit.

Komputer hebat dalam pekerjaan rutin tetapi payah untuk pekerjaan yang memerlukan pemahaman, kreativitas, sementara kita manusia kebalikannya, cepat bosan jika melakukan hal-hal sepele berulang-ulang. Fakta bahwa komputer bagus, hebat untuk perkerjaan yang kita tidak suka, merupakan sebab mengapa kita perlu belajar 'berbicara' kepada komputer dengan cara yang disebut pemrograman.

Ada banyak bahasa pemrograman yang sudah dibuat orang sejak ditemukannya komputer seperti Assembler, FORTRAN, BASIC, LISP, Prolog, Perl, C/C++, Pascal, Java, Javascript, PHP, Swift, Kotlin, Python, dan masih banyak lagi.

Tetapi kenapa Python? Karena:

- bahasa dinamis lebih produktif
- kode Python lebih mudah dibaca
- kode Python lebih mudah perawatannya
- Python memiliki tipe data bawaan tingkat tinggi
- waktu pengembang lebih berharga dari waktu CPU

Dari pengalaman para guru besar seperti Prof. Allen B. Downey, ternyata Python cocok diajarkan sebagai bahasa pengantar karena lebih mudah dicerna, lebih mudah diterima dibanding Java. Fakta bahwa Institut Teknologi Massachusetts, MIT Amerika, mengajarkan Python sebagai pengantar untuk Computer Science CS.6.000 menggantikan bahasa Scheme menggambarkan bahwa Python selain mudah, memiliki logika algoritma yang memadai untuk dikuasai.

Kosakata

Tidak seperti bahasa percakapan biasa, kosakata Python ternyata sangat sedikit, kita menyebutnya 'reserved words', dan setiap kata tersebut hanya mempunyai arti tunggal bagi Python. Python juga berkembang sehingga versi yang lebih baru mungkin memiliki lebih banyak kosakata, tetapi perbedaannya tidak banyak.

False	class	from	or
None	continue	global	pass
True	def	if	raise

and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

Untuk Python versi 3.10 ada tambahan kosakata *match* dan *case* yang sampai saat penulisan ini, belum tercantum dalam daftar kosakata pada perintah help()|keywords.

Interaksi dengan Python

Sebelum mencoba berinteraksi dengan Python, Anda perlu melakukan instalasi Python di komputer Anda terlebih dulu. Jika operating system komputer Anda linux, maka kemungkinan besar python sudah terinstal, siap dipakai. Untuk instruksi instalasi di Windows, Anda dapat merujuk ke www.py4e.com atau situs python <https://www.python.org/ftp/python/3.10.1/python-3.10.1-amd64.exe>

Saya sarankan juga agar Anda menginstal pip, ipython dan jupyter-lab agar Anda dapat mencoba kode program Python dan menuliskan penjelasan sebagai pengingat.

Setelah menginstal python, silahkan Anda masuk ke terminal command line lalu ketikan perintah berikut:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py
pip install ipython
pip install jupyterlab
```

Jika semua berjalan mulus, Anda dapat masuk ke web lokal jupyter lab melalui browser Anda dengan perintah berikut pada terminal command line:

```
jupyter lab
```

In [1]: `print('halo teman')`

```
halo teman
```

In [2]: `2 * 3 * 7`

Out[2]: 42

Selain mengeksekusi deretan kalimat kode/program, shell Python dapat juga berfungsi sebagai kalkulator. Kalkulator bilangan bulat Python dapat menghitung angka sangat besar tanpa muncul error overflow bahkan sampai tak berhingga jika memori komputer Anda dapat menampungnya.

In [3]: `(-80538738812075974)**3 + 80435758145817515**3 + 12602123297335631**3`

Out[3]: 42

Selain dalam Jupyter lab, Anda dapat menjalankan Python di command line, editor IDLE bawaan Python, atau integrated development environment PyCharm.

Dari command line ketik python atau python3,

```
>python
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world!')
hello world!
>>> quit()
>
```

Program kita biasanya disimpan dalam file terpisah dengan ekstensi .py misalnya hello.py

Walaupun dapat dieksekusi langsung, sebenarnya Python memiliki suatu format struktur program yang dianjurkan seperti ini:

```
def main():
    print('saya belajar python')

if __name__ == '__main__':
    main()
```

Kita save script di atas dalam file my_first_script.py lalu bisa kita jalankan dengan:

```
>python my_first_script.py
saya belajar python

>
```

Penulisan script mengikuti aturan bahwa definisi fungsi yang dipanggil didahului penulisannya daripada statement/fungsi yang memanggilnya sehingga ketika Python membaca file, maka definisi yang diperlukan sudah lebih dahulu diketahui sebelum kode yang memanggilnya.

```
>type wrong.py
main() # memanggil fungsi main

def main(): # definisi fungsi main
    print('is it okay?')

>python wrong.py
Traceback (most recent call last):
  File "C:\Users\emang\wrong.py", line 1, in
    main()
NameError: name 'main' is not defined. Did you mean: 'min'?

>
```

Aturan urutan penulisan ini kaku tidak dapat ditawar karena Python tidak memiliki mekanisme deklarasi seperti pada C/C++ dengan header, sebagai interpreter Python membaca baris demi baris

dari atas ke bawah. Demikian sehingga bila terjadi error seperti di atas, Anda dapat mengerti sebabnya dan dengan mudah dapat mengatasinya.

VARIABLES, OBJECT

Di atas kita sudah menggunakan deretan huruf yang disebut string dan nilai angka angka. Untuk dapat menggunakan ulang nilai atau string yang sudah kita tuliskan (secara literal) maka digunakan variabel. Variabel adalah nama yang digunakan untuk menunjuk suatu nilai/string atau secara umum suatu objek. Objek adalah suatu struktur pada lokasi memori tertentu yang mengandung nilai atau string, tipe objek dan beberapa hal penting lainnya.

Untuk mengasosiasikan nama variabel kepada sebuah objek kita menggunakan assignment statement **sebuah_nama = suatu_objek** seperti misalnya

```
umur = 7
hobby = "main game online"
```

Berbeda dengan C/C++, Java, dalam Python variabel bukan nama suatu alokasi memori, tetapi nama pointer atau referens ke suatu objek. Variabel berisi pointer bukan nilai/objek, sehingga sifatnya dinamis, tidak menjadi soal bila dipakai ulang untuk objek lain dengan tipe berbeda, karena semua pointer sama ukuran bytesnya, walaupun objek yang dirujuknya bermacam-macam ukuran dan tipenya.

In [4]:

```
umur = 7
print(umur, type(umur))
umur = "delapan tahun kurang seminggu lagi"
print(umur, type(umur))
```

```
7 <class 'int'>
delapan tahun kurang seminggu lagi <class 'str'>
```

Variabel memegang status objek terkait sekarang yang bisa berubah bila ada assignment baru. Suatu objek tidak berubah, tempat atau nilainya sejak dibuat sampai di daur ulang. Perhatikan contoh berikut dimana ada dua variabel merujuk kepada objek yang sama kemudian ada assignment baru yang terjadi.

In [6]:

```
umur_budi = umur_bambang = 7
print(f'{umur_budi=}, {umur_bambang=}')
umur_budi = 8
print(f'{umur_budi=}, {umur_bambang=}')
```

```
umur_budi=7, umur_bambang=7
umur_budi=8, umur_bambang=7
```

Awalnya ada satu objek instans integer yang dirujuk dua variabel. Saat assignment baru, maka ekspresi sisi kanan tanda = merupakan suatu permintaan pembuatan objek baru dengan nilai berbeda kepada memory manager, lalu setelah dibuatkan objeknya, referensnya, pointer, dicatatkan ulang ke variabel di sisi kiri tanda = dalam hal ini variable umur_budi. Beginilah prinsip kerja Python dengan objek, pointer objek dan variable assignment.

Dalam bahasa lain yang statik, misalnya C, kedua variabel akan berisikan copy nilai yang sama, dan

jika ada perubahan nilai maka perubahannya langsung diterapkan pada lokasi variabel bersangkutan. Nilainya diedit, dimutasi selama masih dalam tipe yang sama sehingga tidak memerlukan perubahan struktur atau jumlah bytesnya. Perubahan tipe variabel tidak diperbolehkan dalam C apalagi Java, tetapi tidak menjadi soal dalam Python.

In [30]:

```
def ppb(a,b):
    "Pembagi Persekutuan terBesar dari dua bilangan, algoritme Euklides"
    while b != 0:
        a, b = b, a%b
    return a
```

Contoh di atas yang menampilkan 'iklan' Python (seperti hello worldnya C), dipresentasikan oleh Guido van Rossum (pembuat Python) pada tahun 2003, sebagai ciri-ciri Python yaitu:

- tanpa deklarasi fungsi ataupun variabel
- blok/grup dengan indentasi dan titik dua
- doc string dokumentasi merupakan bagian dari sintaks fungsi
- assignment paralel (bertukar nama tanpa perantara a,b = b,a)

Dalam Python semua entitas adalah objek, semua struktur data, sederhana atau kompleks, semua fungsi, buatan pengguna atau bawaan adalah objek. Konsistensi ini mempermudah pola pemahaman dan penggunaan komponen pemrograman Python. Semua objek yang dibuat pengguna dikumpulkan, disimpan dalam bagian memori yang disebut heap.

FUNGSI dan Metoda

Istilah fungsi diambil dari istilah matematika, fungsi adalah suatu proses memanipulasi argumen input menjadi suatu output, dengan catatan dalam pemrograman komputer fungsi ada dua jenis yaitu pertama, fungsi murni (*fruitful function*) seperti definisi matematis, yang mengembalikan suatu objek output, dan kedua, fungsi dengan efek samping (*void function*) yang memberikan dampak pada lingkungan tetapi tidak mengembalikan output. Pada contoh di atas, semuanya adalah fungsi efek samping, print() melakukan sesuatu pada console berdasarkan input, dir() menampilkan namespace, demikian juga main() dan fungsi_abc().

Fungsi murni contohnya, pow() mengembalikan nilai yang dipangkatkan.

In [1]:

```
x = 3
y = 2
z = pow(x,y)
print(f'{z=}')
```

z=9

Sebagai tambahan fungsi bawaan, kita dapat mendefinisikan suatu fungsi sendiri sesuai kebutuhan kita. Fungsi didefinisikan dengan keyword def, nama fungsinya dan sepasang tanda kurung untuk menampung argumen input.

In [54]:

```
### recursion
def factorial(n, ans=1):
    if n == 1:
        return ans
```

```

    return factorial(n-1, n*ans)

print(factorial(5))

```

120

Fungsi murni dapat juga didefinisikan tanpa nama, hanya parameter, body dan argumen, dengan keyword lambda yang otomatis mengembalikan nilai ekspresi dari badan fungsi.

In [4]: `(lambda x,y: x**y)(3,2)`

Out[4]: 9

In [10]: `### anagram adalah permainan huruf membentuk kata berbeda
buat fungsi untuk menentukan apakah dua ungkapan masukan adalah anagram atau bukan
from collections import Counter
def anagram_kah(kata1, kata2):
 def cc(k):
 return Counter(k.lower().replace(' ', ''))

 return cc(kata1) == cc(kata2)

print(anagram_kah("Eleven plus Two","Twelve plus One"))`

True

Fungsi dapat didefinisikan secara bertingkat, berjalin, fungsi di dalam fungsi. Counter adalah suatu objek turunan dictionary yang khusus menghitung pemunculan item masukan, dibuat berdasarkan model matematis multiset. Kita akan membahas class dictionary nanti. Dalam contoh di atas, Counter yang diberikan asupan kata kata, secara otomatis menghitung berapa kali huruf per huruf muncul. Outputnya adalah pasangan kunci&nilai (dictionary) dimana kuncinya adalah huruf-huruf yang berbeda dan nilainya adalah jumlah kemunculan dalam kata/ungkapan asupan.

In [25]: `s = "String itu immutable, TIDAK dapat diedit"
print(s.lower())
print(s.lower().replace(' ', ''))
print(s) # s tidak berubah masih seperti aslinya`

```

string itu immutable, tidak dapat diedit
stringituimmutable,tidakdapatdiedit
String itu immutable, TIDAK dapat diedit

```

Fungsi lower() adalah suatu fungsi yang khusus terkait dengan objek tipe string, sehingga kita menyebutnya **metoda** dari string, sama seperti metoda *replace()*. Inputnya bukan berada di dalam kurung tetapi di depannya. Artinya kita memanggil suatu atribut dari objek string *s* yang dapat dieksekusi (callable) yang dalam contoh di atas *lower()* dan *replace()*.

Output dari metoda string adalah suatu objek baru sedangkan objek inputnya tidak berubah.

Berikut metoda string yang selengkapnya.

In [26]: `print([atr for atr in dir(str) if '_' not in atr])`

```

['capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'fin

```

```
d', 'format', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Tidak mengapa bila bila Anda belum memahami command di atas, maksudnya hanya menampilkan semua metoda dari string kecuali metoda-metoda khusus lainnya. Bandingkan dengan eksekusi

```
print(dir(str))
```

yang menampilkan hasil lebih banyak.

Untuk fungsi built in kita dapat melihat daftarnya dalam

```
print(dir(\_\_builtins\_\_))
```

Selain menggunakan fungsi bawaan Python kita mendefinisikan fungsi buatan sendiri dengan tujuan:

- agar dapat dipanggil berulang walau hanya menuliskan sekali saja
- menyembunyikan detil implementasi, kita hanya perlu ingat abstraksinya
- membuat program menjadi terstruktur sehingga lebih mudah dipahami

NONE

Kembali ke pembahasan fruitful function dan void function.

Fungsi *fruitful* mengembalikan sesuatu objek sebagai output, dan fungsi *void* tidak mengembalikan apa-apa atau dengan kata lain, mengembalikan 'bukan apa-apa'.

Python memiliki suatu objek 'bukan apa-apa' yang dinamakan None. Kita akan melihatnya dalam contoh berikut.

```
In [2]: def f():
    pass

x = f()
print(f'{x = }')
print(f'{type(x) = }')

y = print(..bukan apa-apa, None..')
print(f'{y = }')
print(f'{type(y) = }')
```

```
x = None
type(x) = <class 'NoneType'>
..bukan apa-apa, None..
y = None
type(y) = <class 'NoneType'>
```

Fungsi f() tidak memiliki kalimat return dalam definisinya, maka secara default Python akan mengembalikan objek None sebagai return valuenya.

Fungsi print() tidak mengembalikan sesuatu objek sebagai return, maka Python mengembalikan objek None.

None adalah objek real, ada id/alamat-nya, tetapi hanya satu, *singleton* istilahnya, tidak ada copynya.

In [3]:

```
print(id(None))
print(x==y, x is y) # x,y dari contoh sebelumnya None
```

140734242154488

True True

Namespace, Scope

Nama variabel disimpan dalam suatu daftar atau environment yang disebut namespace. Namespace ini tidak hanya satu tetapi bisa ada beberapa namespace yang dibentuk. Ketika suatu blok fungsi dieksekusi maka dalam scope fungsi tersebut ada suatu namespace lokal yang dibentuk untuk mendaftarkan variabel lokal fungsi itu. Jadi ada beberapa namespace antara lain namespace built-in, global, lokal.

Lingkup penamaan di luar fungsi kita sebut lingkup global (global variable's scope), sedangkan di dalam fungsi lingkupnya lokal. Di luar lingkup global ada lingkup yang membungkusnya yaitu lingkup bawaan (built-in). Contoh, fungsi print(), nama fungsi print didaftarkan di lingkup bawaan. Dalam lingkup lokal, kita dapat membaca objek lingkup global atau lingkup bawaan, tetapi tidak dapat mengubah nilainya. Ketika kita mencoba melakukan assignment maka otomatis Python akan membuatkan variabel lokal dengan nama itu dan menutup akses ke lingkup di atasnya.

In [55]:

```
def fungsi_abc():
    def fungsi_sub():
        nonlocal namaku
        namaku = 'Bejo Tinurut'
        ujian_akhir = 90
        print('mengubah variabel "non-lokal"')
        print(dir(),namaku)
        return ujian_akhir
    namaku = 'Bejo'
    nilai_ujian = 85
    print(dir(), namaku, nilai_ujian)
    nilai_ujian = fungsi_sub()
    print(namaku, nilai_ujian)

def main():
    namaku = 'Bento'
    print(dir(), namaku)
    fungsi_abc()

namaku = 'variabel global'
print(namaku)
main()
```

```
variabel global
['namaku'] Bento
['fungsi_sub', 'namaku', 'nilai_ujian'] Bejo 85
mengubah variabel "non-lokal"
['namaku', 'ujian_akhir'] Bejo Tinurut
Bejo Tinurut 90
```

Mendefinisikan variabel pada lingkup global tidak disarankan dan dianggap sebagai praktek berbahaya. Agar penamaan (namespace) terkendali sebaiknya semua variabel berada di dalam fungsi. Transfer variabel antar fungsi yang disebut passing parameter dilakukan dengan argumen dan return value.

Dalam situasi dimana diperlukan akses read/write langsung, maka dapat ditempuh strategi penempatan fungsi di dalam fungsi lalu untuk menyatakan bahwa variabel yang akan diakses itu berada di luar fungsi sebelah dalam digunakan statement **nonlocal \<namavar>**.

DESAIN ANTAR MUKA (INTERFACE)

Pada bagian ini kita akan mempelajari proses mendesain fungsi supaya bekerja dengan baik. Disini kita akan menggunakan pustaka modul *turtle*, yang melaluiinya kita dapat membuat beberapa grafik sebagai sarana praktek. Modul turtle sudah disertakan dalam instalasi Python dan kode latihan kita akan dijalankan dalam konsol shell Python.

bagian ini mengambil materi dari Think Python, Allen B. Downey chapter 4

Modul turtle

Kita dapat memeriksa apakah modul turtle tersedia dengan perintah di shell Python (setelah memanggil python dari command console) sebagai berikut:

```
>>> import turtle
>>> pena = turtle.Turtle()
```

Jika sudah tersedia maka akan muncul window baru dengan tanda panah kecil di tengah. Tutup kembali windownya untuk saat ini.

Salin kode berikut ke dalam file bernama poligon.py

```
import turtle
pena = turtle.Turtle() # membuat instans dari turtle
print(f'{pena = }')
pena.fd(100)
pena.lt(90)
pena.fd(100)
turtle.mainloop() # melepas kontrol ke window
```

lalu jalankan dari command console dengan perintah

```
> python poligon.py
```

Dalam console akan ditampilkan bahwa pena adalah suatu objek tipe turtle yang selanjutnya akan kita gunakan dengan metoda-metoda yang terkait dengan objek tersebut. Metoda class *turtle.mainloop()* adalah metoda untuk menampilkan window grafik agar tidak tertutup langsung karena kode kita sudah mencapai baris terakhir, dan selanjutnya kita bisa menutup window dari elemen window itu sendiri (X).

Penjelasan tentang kelas *turtle* dapat dipelajari dari dokumentasinya di <https://docs.python.org/3.10/library/turtle.html> atau dari shell python ketik

```
>>> import turtle
>>> dir(turtle)
>>> help(turtle)
```

Metoda yang akan kita gunakan dalam praktik kita adalah terutama maju/forward(fd), dan belok kiri/ left(lt). Metoda yang lain dapat dilihat dari perintah di atas.

Perintah maju fd(jarak) akan menorehkan garis sepanjang jarak. Nilai jarak dapat berupa bilangan bulat maupun pecahan. Perintah belok kiri lt(sudut) akan mengarahkan panah ke arah sudut yang diberikan. Contoh di atas, pena.fd(100); pena.lt(90); pena.fd(100) menggambar dua potong garis ke arah timur berbelok ke arah utara.

Edit poligon.py dengan memakai perintah fd(100) lt(90) sebanyak empat kali untuk membuat suatu bujur sangkar. Silahkan selesaikan latihan ini sebelum Anda meneruskan membaca.

loop

Anda mungkin telah menuliskan perintah menggambar bujur sangkar seperti ini,

```
pena.fd(100)
pena.lt(90)
```

```
pena.fd(100)
pena.lt(90)
```

```
pena.fd(100)
pena.lt(90)
```

```
pena.fd(100)
</pre>
```

Kita dapat menuliskan secara lebih ringkas dengan menggunakan perintah perulangan dalam suatu blok seperti ini,

```
for i in range(4):
    pena.fd(100)
    pena.lt(90)
```

Kalimat *for* disebut juga loop atau **for loop** karena mengulangi perintah dalam blok di bawahnya, 4 kali dalam contoh kita.

Latihan

- Buatkan suatu fungsi bernama bujursangkar dengan parameter masukan t sebagai objek turtle. Fungsi itu membuat gambar bujursangkar menggunakan turtle tersebut. Buat pemanggil fungsi dengan argumen pena kepada fungsi bujursangkar itu.

2. Tambahkan parameter lain bernama jarak ke dalam fungsi bujursangkar, dan ubah badan fungsi agar menggunakan parameter jarak untuk panjang sisi sisinya. Buat pemanggil fungsi dengan tambahan argumen jarak. Jalankan program beberapa kali dengan nilai jarak yang berbeda.
3. Salin fungsi bujursangkar dan namai poligon lalu tambahkan parameter bernama n. Ubah badan fungsi sehingga menggambar poligon segi-n.
Petunjuk: sudut luar yang digunakan adalah $360/n$ derajat.
4. Buatkan suatu fungsi bernama lingkaran dengan parameter masukan t turtle, r radius, yang menggambar lingkaran pendekatan dengan memanggil fungsi poligon dengan argumen jarak, dan segi-n yang sesuai. Periksa fungsi Anda dengan beberapa nilai radius.
Petunjuk: hitung keliling lingkaran dan pastikan $\text{jarak} * n = \text{keliling lingkaran}$.

Jawaban

Berikut adalah jawaban latihan di atas, tetapi sebaiknya Anda mencoba menyelesaikan sendiri latihan di atas sebelum membandingkannya dengan jawaban berikut ini.

1. Enkapsulasi

```
import turtle
def bujursangkar(t):
    for _ in range(4):
        t.fd(100)
        t.lt(90)

pena = turtle.Turtle()
bujursangkar(pena)
turtle.mainloop()
```

Membungkus sepotong kode ke dalam fungsi dinamakan **enkapsulasi**. Salah satu manfaat enkapsulasi ialah bahwa kode yang dibungkus sekarang dapat memiliki nama yang memberikan rangkuman apa yang dijalankan di dalamnya. Manfaat lain ialah bahwa fungsi tersebut dapat dipanggil ulang dengan argumen yang sama atau berbeda, yang lebih ringkas daripada harus menyalin seluruh badan fungsi berulang ulang.

2. Generalisasi

Salin kode berikut ke file bujursangkar.py dan jalankan pada command console.

```
import turtle

def bujursangkar(t, jarak):
    "bujursangkar dengan turtle"
    for _ in range(4):
        t.fd(jarak)
        t.lt(90)

pena = turtle.Turtle() # membuat instans dari turtle
for i in range(12):
    bujursangkar(pena, 100+ i*i)
    pena.lt(-30)
    pena.fd(10)
```

```
        turtle.mainloop() # melepas kontrol ke window
```

Menambah parameter pada fungsi membuat fungsi tersebut dapat dipergunakan secara umum, disebut generalisasi. Jika pada contoh sebelumnya ukuran bujursangkar konstan ditentukan dari dalam fungsi, maka dengan tambahan parameter dapat diatur dari pemanggil fungsi.

3. Generalisasi (lagi)

Dalam fungsi bujursangkar, sudutnya sudah ditentukan dari dalam fungsi yaitu 90 derajat, artinya jumlah sisinya selalu konstan segi-4. Fungsi poligon menambah parameter jumlah segi/sisi sehingga dapat ditentukan dari pemanggil. Dengan bertambahnya argumen maka perlu digunakan nama supaya mudah diingat dan bebas letak urutan argumennya.

```
import turtle

def setposisi(t,x,y):
    "set posisi awal"
    t.penup()
    t.setx(x)
    t.sety(y)
    t.pendown()

def poligon(t, sisi, segi):
    "poligon segi-n"
    sudut = 360 / segi
    for _ in range(segi):
        t.forward(sisi)
        t.left(sudut)

pena = turtle.Turtle() # membuat instans dari turtle
setposisi(pena,-100,100)
poligon(pena, 70, 4)

setposisi(pena,50,50)
poligon(pena, segi=5, sisi=100) # keyword arguments

setposisi(pena,-160,-130)
poligon(pena,60, 9)

turtle.mainloop() # melepas kontrol ke window
```

4. Desain antarmuka (interface)

Untuk membuat lingkaran kita dapat menggunakan fungsi poligon dengan segi-n yang banyak, misalnya 50. Jika r adalah radius lingkaran, maka keliling lingkaran $2\pi r$ adalah panjang sisi kali segi-n. Maka jika kita tetapkan segi n = 50, maka panjang sisi = keliling/50.

```
def lingkaran(t,r):
    pjg_sisi = 2*math.pi*r / 50
    poligon(pena, segi=50, sisi=pjg_sisi)
```

Membuat jumlah segi $n = 50$ secara konstan bisa bermasalah, mungkin kurang untuk radius yang besar dan terlalu boros untuk radius yang kecil, sehingga mungkin kita berpendapat bahwa sebaiknya n dijadikan parameter juga. Tetapi hal itu akan membuat interface menjadi kurang rapi. Fungsi `lingkaran(t, radius, n)` menjadi terasa kurang pas bukan? Solusinya agar interface tetap rapi nilai n harus kita buatkan sebanding dengan radius, makin besar radius makin banyak n . Jadi misalnya kita rumuskan, $n = 3 + \text{int}(\text{keliling}/5)$, minimal 3 dan bilangan bulat. Faktor pembagi dapat diatur sesuai kualitas yang diinginkan tetapi kualitas tersebut akan terlihat sama untuk radius kecil atau radius besar.

```
def lingkaran(t, radius):
    keliling = 2 * math.pi * radius
    segi = 3 + int(keliling/5)
    pjl_sisi = keliling / segi
    polygon(pena, segi=segi, sisi=pjl_sisi)
```

5. Refactoring

Berikut contoh kode yang diperbaiki struktur programnya, *refactoring* istilahnya. Salin dan jalankan dalam command console dan nikmati pergerakannya dalam slow motion, namanya juga turtle, kura-kura grafik.

```
In [ ]:
import turtle
import math

def polyline(t, n, length, angle):
    for i in range(n):
        t.fd(length)
        t.lt(angle)

def polygon(t, n, length):
    angle = 360 / n
    polyline(t, n, length, angle)

def arc(t, r, angle):
    arc_length = 2*math.pi*r *angle /360
    n = 1 + int(arc_length/4)
    step_length = arc_length /n
    step_angle = float(angle)/n
    polyline(t,n, step_length, step_angle)

def circle(t,r):
    arc(t,r,360)

def setposisi(t,x,y):
    "set posisi awal"
    t.penup()
    t.setx(x)
    t.sety(y)
    t.pendown()

def bunga_A(t, k):
    for _ in range(k):
        arc(t, 100, 360/k)
        t.lt(180 - 360/k)
```

```
arc(t, 100, 360/k)
t.lt(180)

def bunga_B(t,k):
    for _ in range(k):
        arc(t, 100, 4*360/k)
        t.lt(180 - 2*360/k)

def bunga_C(t,k):
    for _ in range(k):
        arc(t, 100, 8*360/k)
        t.lt(180 - 4*360/k)

def bunga_D(t,k):
    for _ in range(k):
        arc(t, 100, 3*360/k)
        t.lt(180 - 1.5*360/k)

pena = turtle.Turtle() # membuat instans dari turtle

setposisi(pena, 30, 80)
bunga_A(pena,7)

setposisi(pena,-55,-200)
bunga_D(pena,7)

setposisi(pena, 140, 0)
bunga_B(pena,10)

setposisi(pena,-300, 0)
bunga_C(pena,18)

turtle.mainloop() # meLepas kontrol ke window
```

Latihan dan contoh di atas walaupun sebagai kegiatan pemrograman yang menyenangkan diharapkan Anda dapat memetik beberapa pelajaran dari padanya, tentang enkapsulasi, generalisasi, interfacing dan refactoring.

In []: