Pemecahan masalah sederhana dan ilustrasinya dengan Python

https://github.com/dudung/sk5003-02-2022-2

Sparisoma Viridi

Master Program in Computational Science, Nuclear Physics and Biophysics Research Division, Department of Physics, Faculty of Mathematics and Natural Sciences, Institut Teknologi Bandung, Bandung 40132, Indonesia

20230401-v2 | https://doi.org/10.5281/zenodo.7790483

Kerangka

Tautan

Tautan

- Materi-materi yang diberikan dapat dilihat pada beberapa tautan berikut

  https://github.com/dudung/py-jupyter-nb

  https://github.com/dudung/python

  https://github.com/dudung/sk5003-02-2022-2

SAP dan referensi

Referensi

Reference 9780367575533

Jose M. Garrido, "Introduction to Computational Models with Python", Routledge, 1st edition, 2020.

Satuan Acara Perkuliahan

Minggu	Topik	Subtopik
1	Pendahuluan	Contoh program Python sederhana dan pemecahan masalah dengan bantuan Python
2	Prinsip-prinsip pemrograman dasar dengan Python	Fungsi, modul, dan struktur program
3	Prinsip-prinsip pemrograman dasar dengan Python	Seleksi dan repetisi
4	Struktur data, orientasi objek, rekursi dalam Python	List Python, string, sekuensi data

Minggu 1

Minggu	Topik	Subtopik	Capaian Belajar
1	Pendahuluan	Contoh program Python sederhana dan pemecahan masalah dengan bantuan Python	Kemampuan untuk memahami dan menguasai contoh program Python sederhana dan pemecahan masalah dengan bantuan Python

R1

C1

- Computer problem solving
- Elementary concepts
- Developing computational models
- Modular design

C2

- Computing with Python
- Data definition
- Simple program
- Temperature conversion
- Program general structure
- Simple function

Minggu 2

Minggu	Topik	Subtopik	Capaian Belajar
2	Prinsip-prinsip pemrograman dasar dengan Python	Fungsi, modul, dan struktur program	Kemampuan untuk memahami dan menguasai fungsi, modul, dan struktur program dalam Python

R1

C3

- Modular decomposition
- Functions
- Built-in functions

C4

- Algorithms
- Flowcharts
- Pseudo-code
- Design structures

Minggu 3

Minggu	Topik	Subtopik	Capaian Belajar
3	Prinsip-prinsip pemrograman dasar dengan Python	Seleksi dan repetisi	Kemampuan untuk memahami dan menguasai seleksi dan repetisi dalam Python

R1

C5

- Conditional expressions
- Logical operators
- Selection structures
- Quadratic formula ($a \neq 0$)

C6

- While loop
- Repeat-until loop
- For loop

Minggu 4

Minggu	Topik	Subtopik	Capaian Belajar
4	Struktur data, orientasi objek, rekursi dalam Python	List Python, string, sekuensi data	Kemampuan untuk memahami dan menguasai list, string, sekuensi data dalam Python

R1

C7

- List
- List of list
- Tuple
- Dictionary
- String

Pendahuluan

Pendahuluan

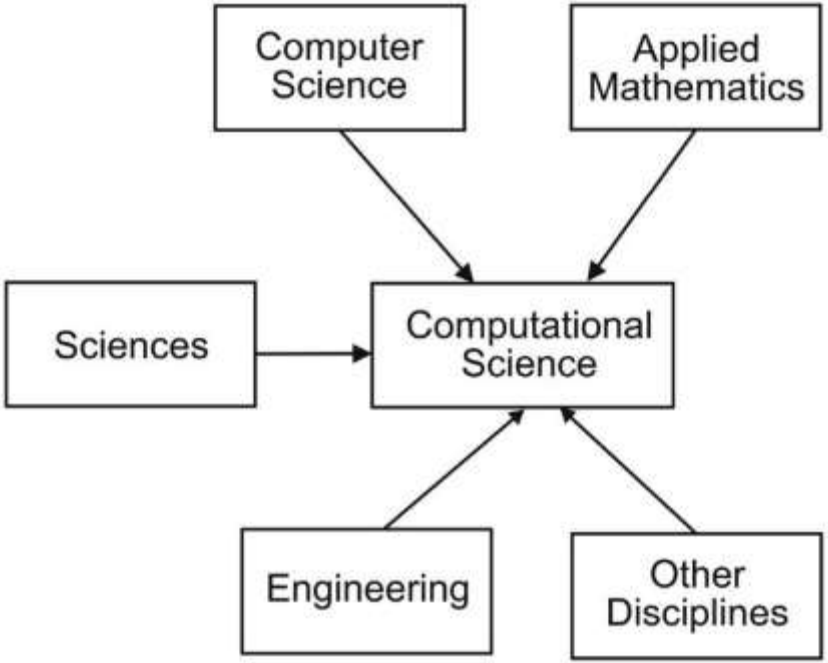
- Computer problem solving
- Elementary concepts
- Developing computational models
- Modular design

Computer problem solving

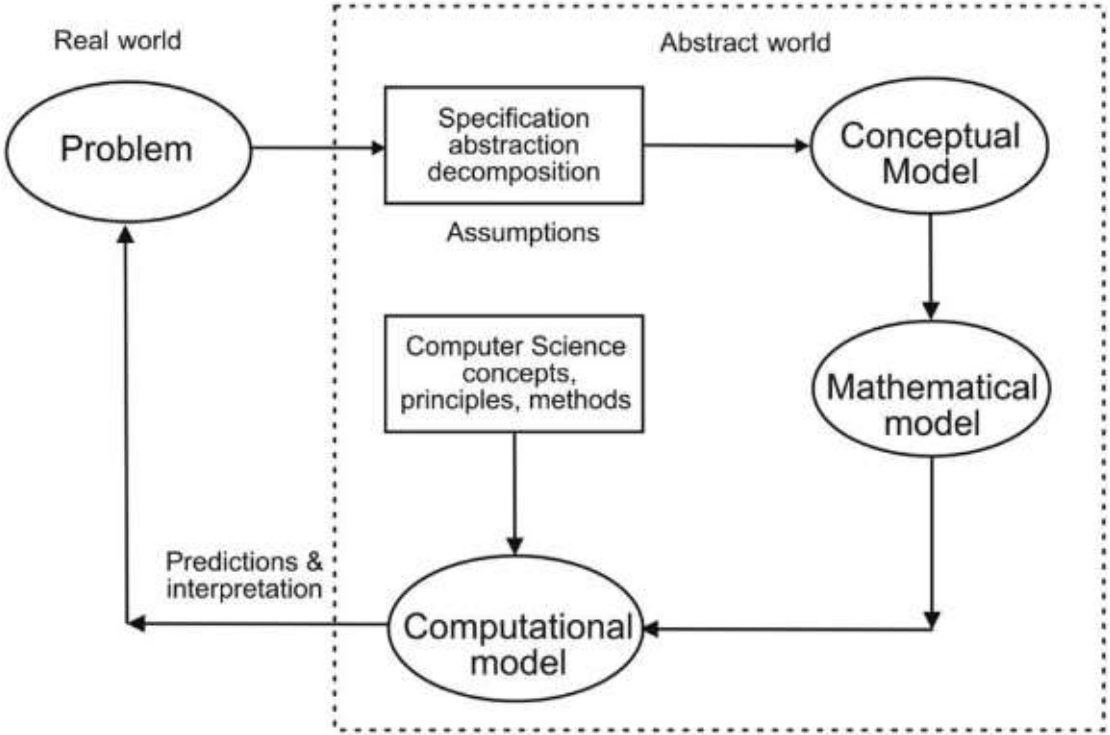
Problem solving is the process of developing a computer solution to a given real-world problem. The most challenging aspect of this process is discovering the method to solve the problem. This method of solution is described by an algorithm. A general process of problem solving involves the following steps:

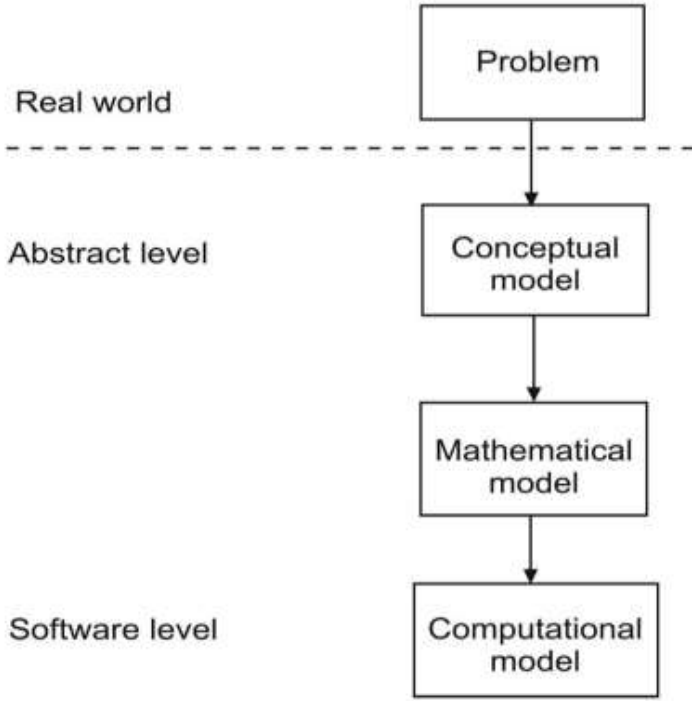
1. Understand the problem.
2. Describe the problem in a clear, complete, and unambiguous form.
3. Design a solution to the problem (algorithm).
4. Develop a computer solution to the problem.

Elementary concepts

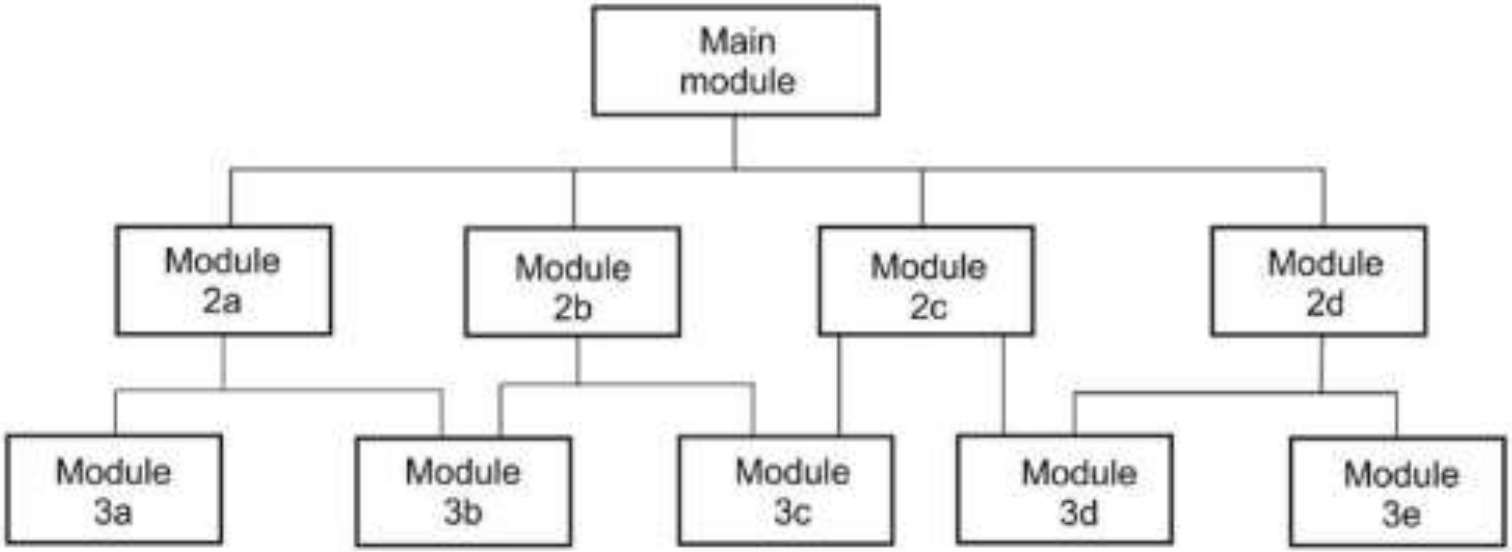


Developing computational models





Modular design



Program sederhana

Program sederhana

- Computing with Python
- Data definition
- Simple program
- Temperature conversion
- Program general structure
- Simple function

Computing with Python

There two modes of computing with Python:

- *Interactive mode*, which provides the Python prompt indicated by `>>>`. Each Python command is directly entered and the Python interpreter responds immediately to the command.
- *Script mode*, which is a sequence of Python commands in a file with a `.py` extension. This file is passed to the Python interpreter to process and is known as a *script*. A typical Python program is edited and stored as a script and an appropriate text editor is used to build a program as a Python script.

Data definition

The data in a program consists of one or more data items. These are manipulated or transformed by the computations (computer operations). In Python, each data definition is specified by assigning a value to a variable and has:

- a reference, which is a *variable* with a unique *name* to refer to the data item, and
- a *value* associated with it.

The name of the reference (variable) to a data item is an *identifier* and is defined by the programmer; it must be different from any *keyword* in the programming language.

Simple program

```
# This script computes 75% of the value of y
y = input ("Enter initial value of y: ")
print "Initial value of y: ", y
y = y * 0.75
print "Final value of y: ", y
```

```
$ python prog03.py
Enter initial value of y: 34.5
Initial value of y:  34.5
Final value of y:  25.875
```

Temperature conversion

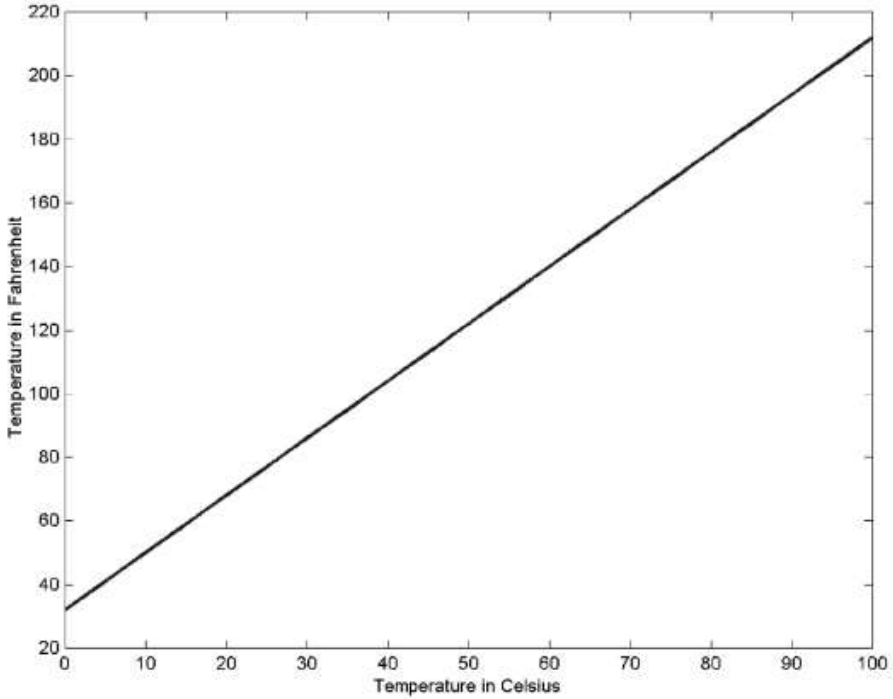
The mathematical representation of the solution to the problem, the formula expressing a temperature measurement F in Fahrenheit in terms of the temperature measurement C in Celsius is:

$$F = \frac{9}{5} C + 32.$$

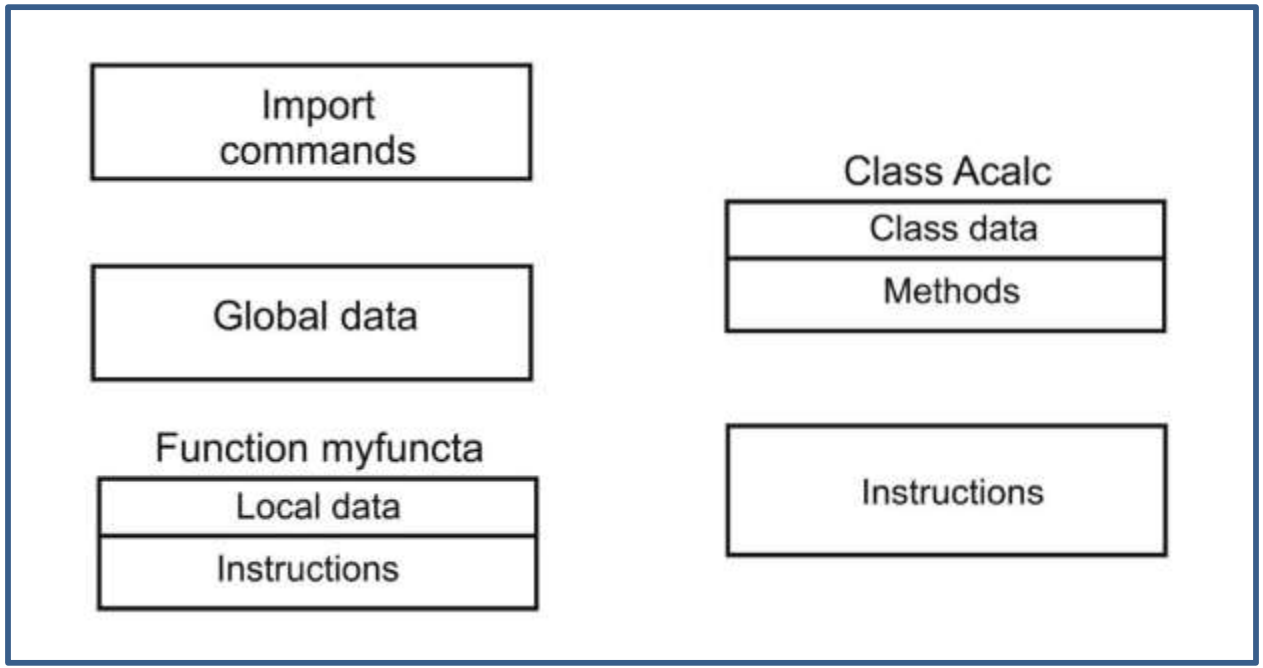
```
$ python tconvctof.py
Enter value of temp in Celsius: 25.0
Value of temp in Celsius:  25.0
Temperature in Fahrenheit:  77.0
```

```
1 """
2    Program     : tconvctof.py
3    Author      : Jose M Garrido
4    Date        : 5-12-2014
5    Description : Read value of temperature Celsius from
6    console, convert to degrees Fahrenheit, and display
7    value of this new temperature value on the output
8    console */
9  """
10
11 C = input("Enter value of temp in Celsius: ")
12 F = C * (9.0/5.0) + 32.0     # temperature in Fahrenheit
13 print "Value of temperature in Celsius: ", C
14 print "Temperature in Fahrenheit: ", F
```

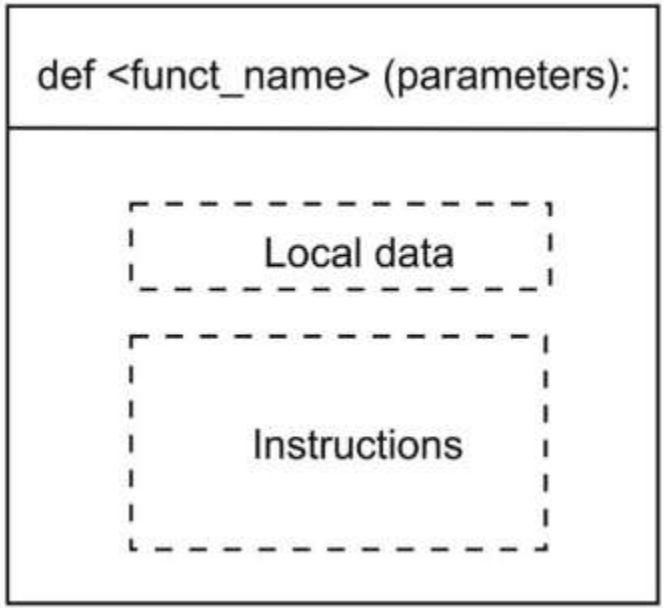
Celsius	5	10	15	20	25	30	35	40	45
Fahrenheit	41	50	59	68	77	86	95	104	113



Program general structure



Simple function




```
def function_name ( [parameters] ) :
      [ local declarations ]
      [ executable language statements ]
```

```
def show_message () :
    """
      This function displays a message
      on the screen.
    """
    print("Computing data")
```

```
 2 # Program     : shmessp.py
 3 # Author      : Jose M Garrido, May 28 2014.
 4 # Description : Define and call a simple function.
 5
 6 def show_message():
 7     """
 8       This function displays a message
 9       on the screen
10     """
11     print "Computing results ..... "
12
13 y = input("Enter a number: ")
14 sqy = y * y
15 show_message()
16 print "square of the number is: ", sqy
```

```
$ python shmessp.py
Enter a number: 12
Computing results .....
square of the number is:  144
```

Dekomposisi / fungsi

Dekomposisi / fungsi

- Modular decomposition
- Functions
- Built-in functions

Modular decomposition

A problem is often too large and complex to deal with as a single unit. In problem solving and algorithmic design, the problem is partitioned into smaller problems that are easier to solve. The final solution consists of an assembly of these smaller solutions. The partitioning of a problem into smaller parts is known as *decomposition*. These small parts are sometimes known as *modular units*, which are much easier to develop and manage.

System design usually emphasizes modular structuring, also called modular decomposition. With this approach, the solution to a problem consists of several smaller solutions corresponding to each of the subproblems. A problem is divided into smaller problems (or subproblems), and a solution is designed for each subproblem. These modular units are considered building blocks for constructing larger and more complex algorithms.

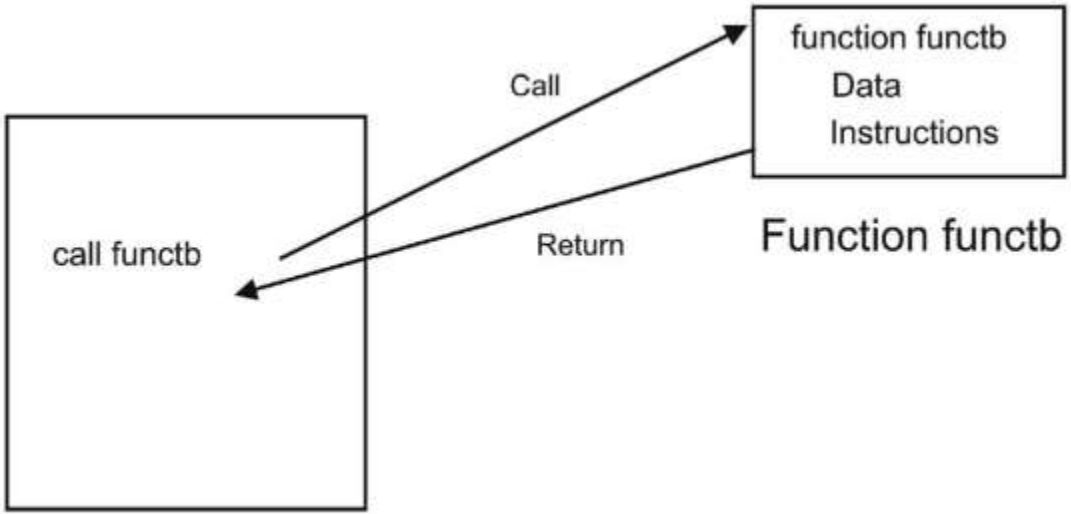
Functions

A Python program is often *decomposed* into modules, and these are divided into classes and functions. A function carries out a specific task in a program.

The data in a function is known only to that function—the scope of the data is *local* to the function. The local data in a function has a limited lifetime; it only exists during execution of the function.

A Python program typically consists of functions and instructions that call or invoke the various functions. In the source code, the general syntactical form of a function definition in the Python programming language is written as follows:

```
def function_name ( [parameters] ) :
    [ local declarations ]
    [ executable language statements ]
```


function functb
Data
Instructions
Call
call functb
Return
Function functb

1. Simple functions, which do not allow data transfer when they are called. The previous example, function *show_message*, is a simple function because there is no data transfer involved.

2. Functions that return a single value after completion.

3. Functions that specify one or more parameters, which are data items transferred to the function.

4. Functions that allow data transfers in both directions. These functions specify one or more parameters and return a value to the calling function.

Built-in functions

Built-in Functions

A	**E**	**L**	**R**
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	**F**	**M**	
ascii()	filter()	map()	**S**
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	**G**	**N**	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	**H**	object()	
callable()	hasattr()	oct()	**T**
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		**P**	**V**
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	**Z**
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		_
	iter()		__import__()

https://docs.python.org/3/library/functions.html

Algorithm, flowchart, pseudo-code

Algorithm, flowchart, pseudo-code

- Algorithms
- Flowcharts
- Pseudo-code
- Design structures

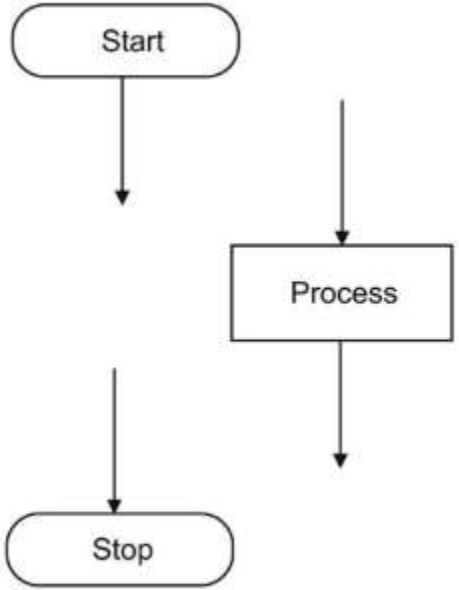
Algorithms

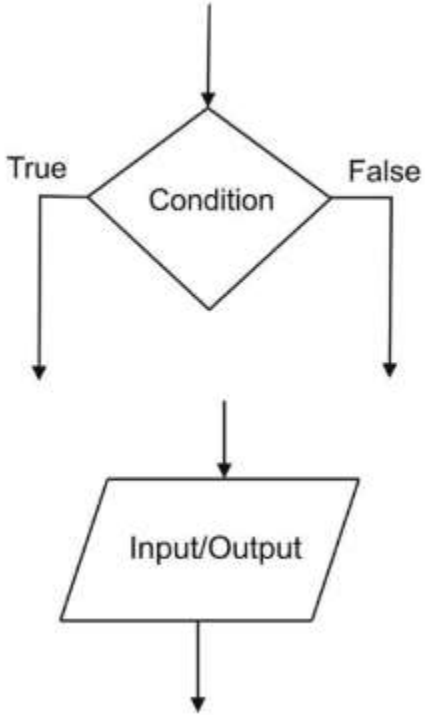
An algorithm is a clear, detailed, precise, and complete description of the sequence of steps to be performed in order to produce the desired results. An algorithm can be considered the transformation on the given data and involves a sequence of commands or operations that are to be carried out on the data in order to produce the desired results.

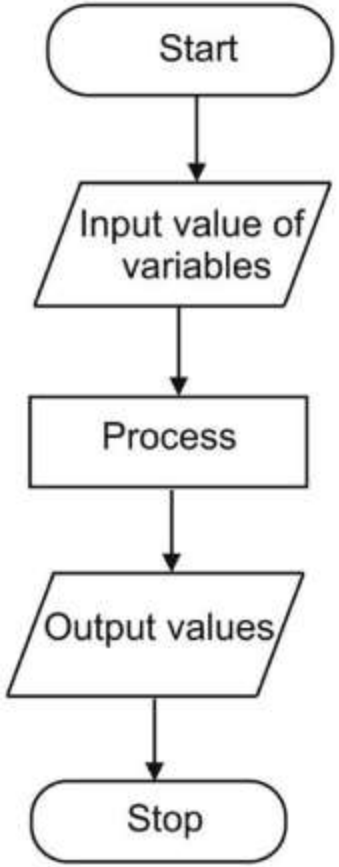
Several notations are used to describe an algorithm. An algorithmic notation is a set of general and informal rules used to describe an algorithm. Two widely used notations are:

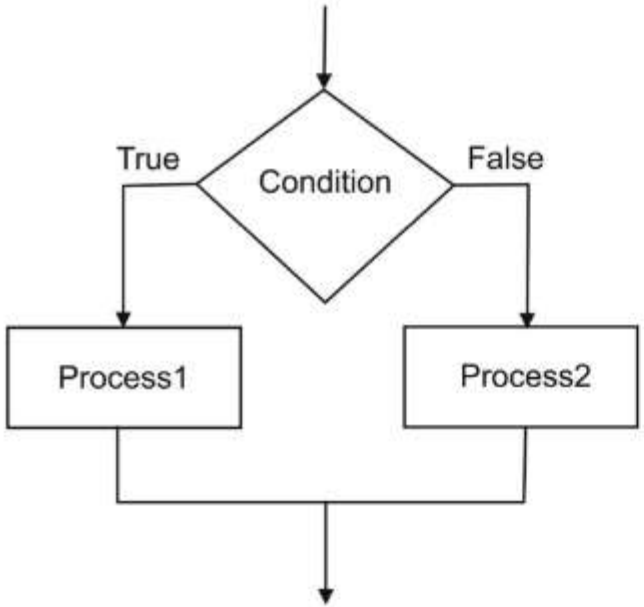
- Flowcharts
- Pseudo-code

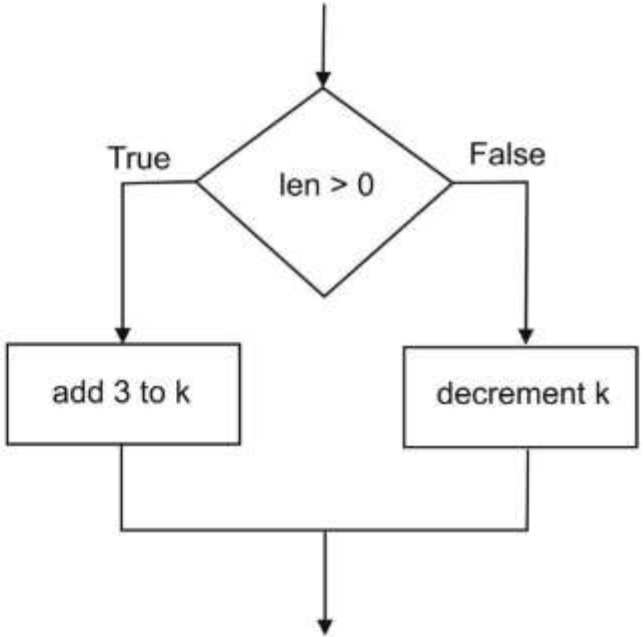
Flowcharts

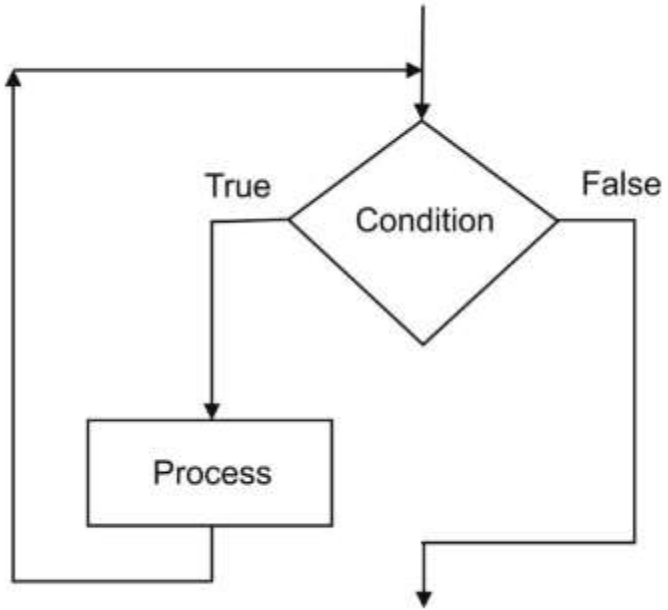

Start
Process
Stop

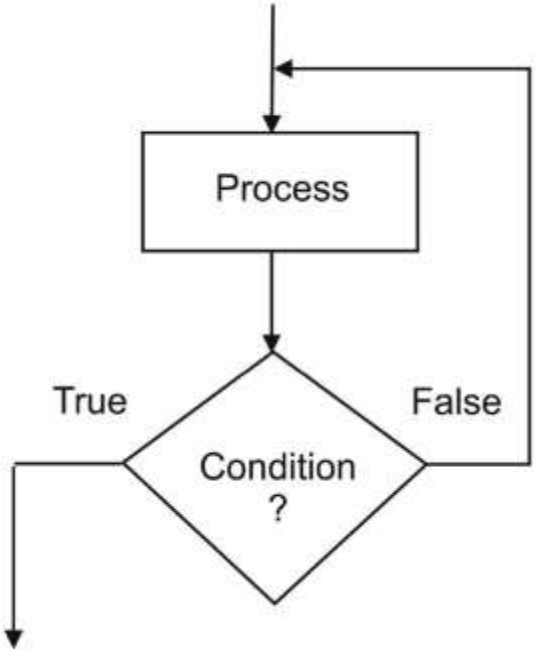

True
Condition
False
Input/Output


Start
Input value of variables
Process
Output values
Stop


True
Condition
False
Process1
Process2


True
len > 0
False
add 3 to k
decrement k





while loop and repeat-until loop

Design structures

The flow of control of an algorithm can be completely defined with only four fundamental design structures. These structures can be specified using flowcharts and/or pseudo-code notations. The design structures are:

1. *Sequence*: Describes a sequence of operations.
2. *Selection*: This part of the algorithm takes a decision and selects one of several alternate paths of flow of actions. This structure is also known as alternation or conditional branch.
3. *Repetition*: This part of the algorithm has a sequence of steps that are to be executed zero, one, or more times.
4. *Input-output*: The values of variables are read from an input device (such as the keyboard) or the values of the variables (results) are written to an output device (such as the screen).

Pseudo-code

Pseudo-code is an informal notation that uses a few simple rules and English for describing the algorithm that defines a problem solution. It can be used to describe relatively large and complex algorithms. It is relatively easy to convert the pseudo-code description of an algorithm to a computer implementation in a high-level programming language.

Kondisi dan seleksi

Kondisi dan seleksi

- Conditional expressions
- Logical operators
- Selection structures
- Quadratic formula ($a \neq 0$)

Conditional expressions

In Python, the relational operators are used with the following notation:

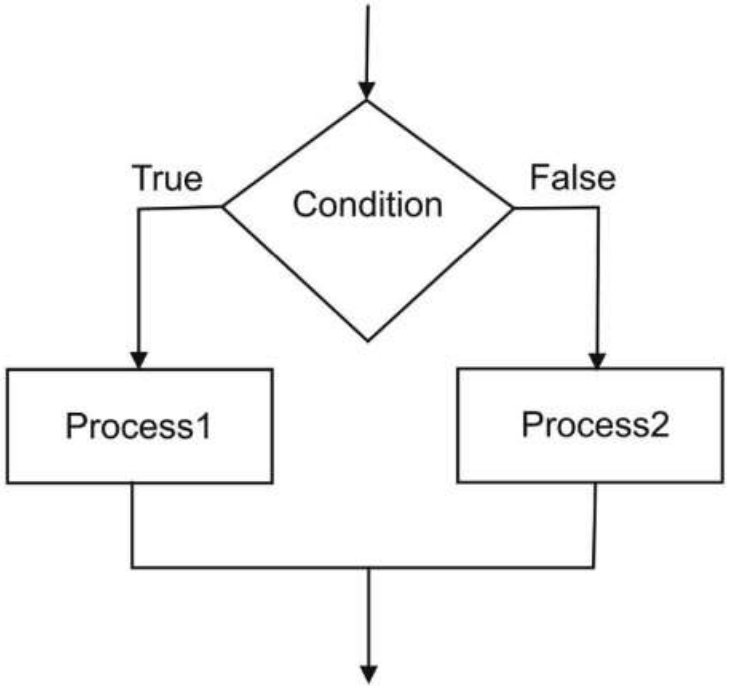
Relational Operator	Description	Arithmetic Notation
>	Greater than	$>$
<	Less than	$<$
==	Equal to	$=$
>=	greater than or equal to	$\geq$
<=	Less than or equal to	$\leq$
!=	Not equal to	$\neq$

Logical operators

cond	**not** cond
True	False
False	True

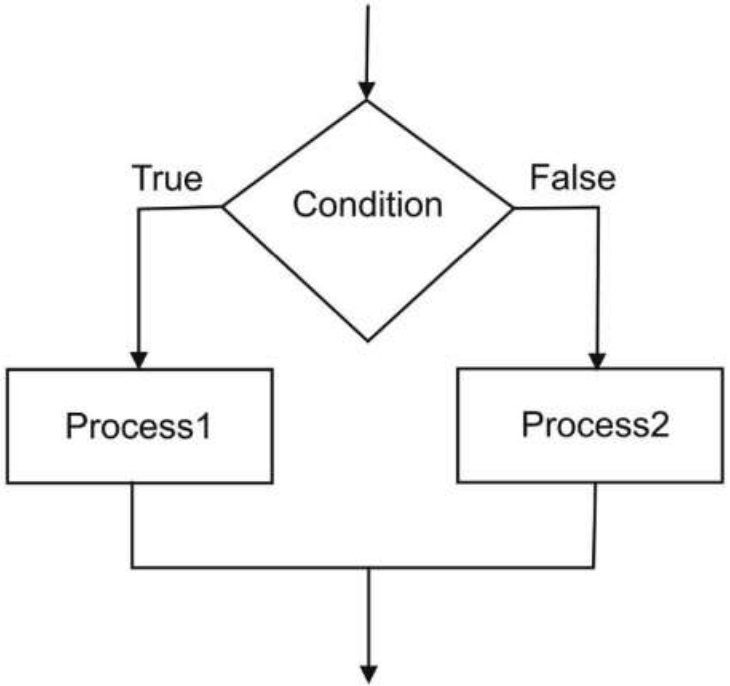
cond1	cond2	cond1 **and** cond2	cond1 **or** cond2
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Selection structures




```
if ⟨ condition ⟩ :
        ⟨ statements in Process1 ⟩
else :
        ⟨ statements in Process2 ⟩
```

```
if (len > 0) :
        k= k + 3
else :
        k=k - 1
```




```
if ⟨ condition ⟩ :
        ⟨ statements in Process1 ⟩
else :
        ⟨ statements in Process2 ⟩
```

```
if a < b or x >= y :
        a = x + 23.45
else :
        a = y
```

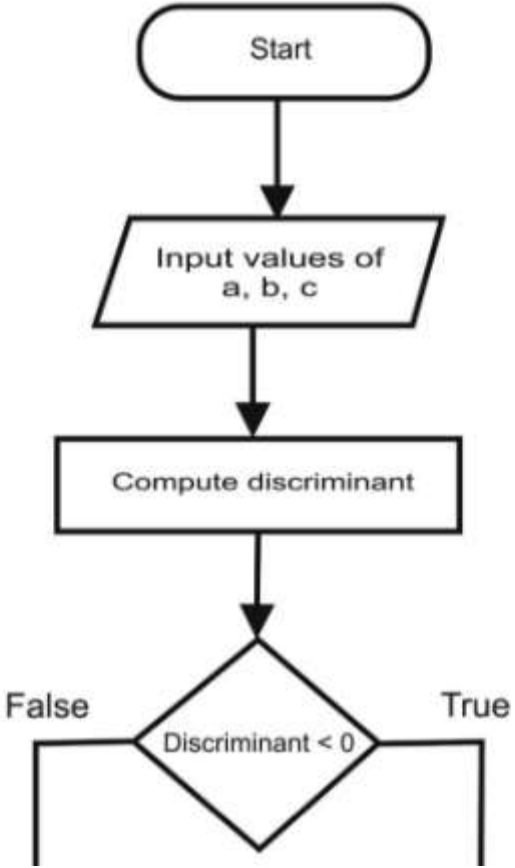
Quadratic formula (a ≠ 0)

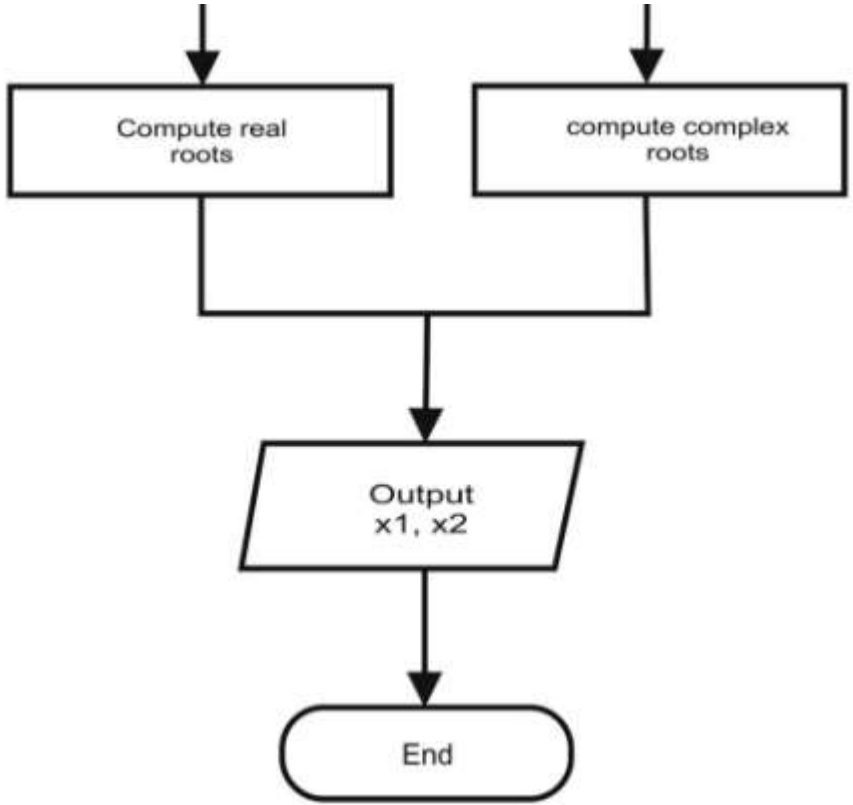
$$ax^2 + bx + c = 0.$$

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

```
Input the values of coefficients  a,  b, and  c
Calculate value of the discriminant
if the value of the discriminant is less than zero
     then calculate the two complex roots
     else calculate the two real roots
endif
display the value of the roots
```

```
read the value of a from the input device
read the value of b from the input device
read the value of c from the input device
compute the discriminant, disc = b^2 - 4ac
if discriminant less than zero
then
        // roots are complex
        compute x1 = (-b + sqrt(disc))/2a
        compute x2 = (-b - sqrt(disc))/2a
else
        // roots are real
        compute x1 = (-b + sqrt(disc))/2a
        compute x2 = (-b - sqrt(disc))/2a
endif
display values of the roots: x1 and x2
```


Start
Input values of
a, b, c
Compute discriminant
False
Discriminant < 0
True

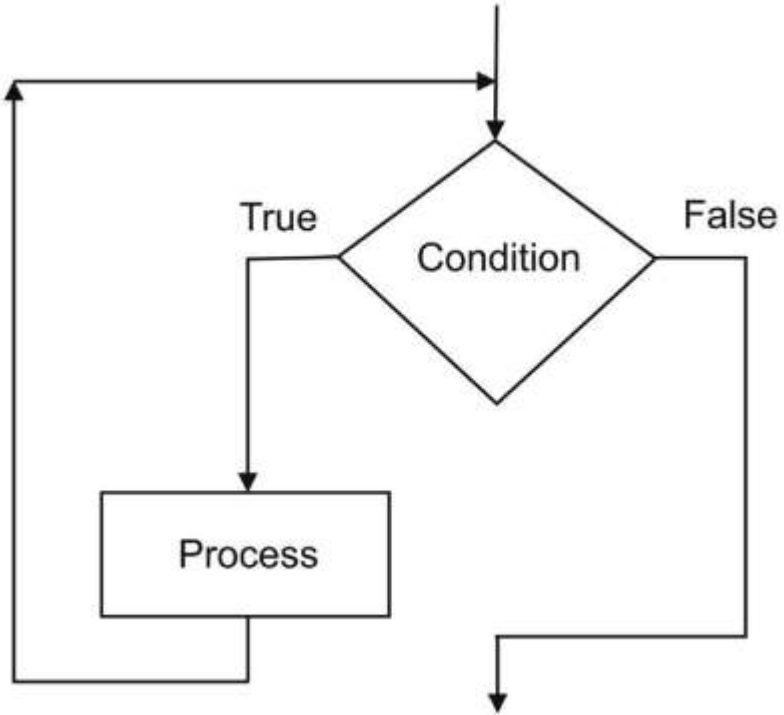

Compute real
roots
compute complex
roots
Output
x1, x2
End

Loop

Loop

- While loop
- Repeat-until loop
- For loop

While loop

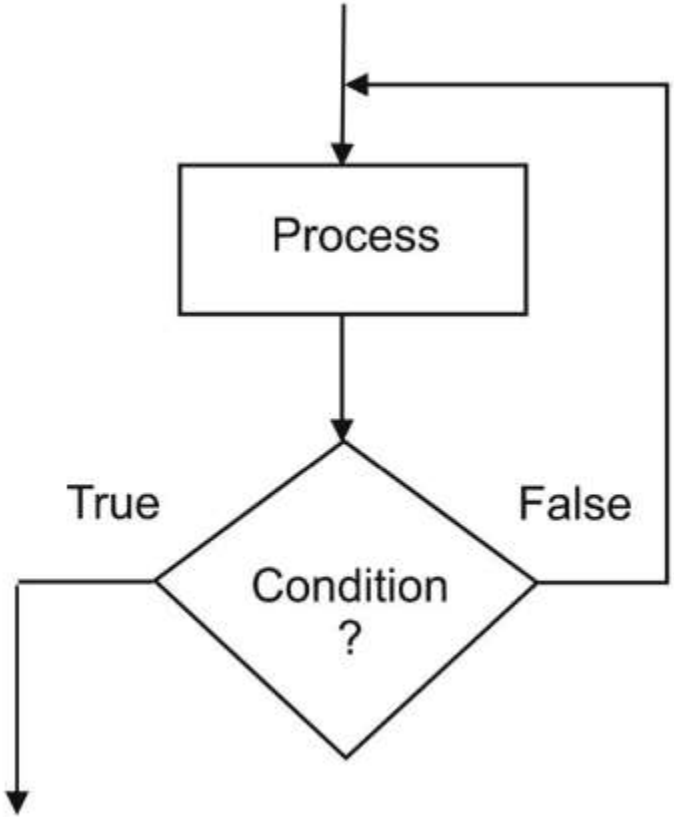


while ⟨ *condition* ⟩ :
⟨ *block of statements* ⟩

```
1 # Script for testing while-loop
2 x = 12.35
3 MAX_NUM = 15
4 j = 0
5 sum = 0.0
6 while ( j <= MAX_NUM)  :
7       sum = sum + 12.5
8       y = x * 2.5
9       j = j + 3
10 print 'Value of sum: ', sum
```

```
$ python test2.py
Value of sum:  75.0
```

Repeat-until loop



not supported in Python

For loop

```
 1 # Script: test6.py
 2 # This script tests a for-loop
 3 x = 3.45
 4 num = 10
 5 sum = 0.0
 6 for j in range(1, num) :
 7         sum = sum + 12.5
 8         y = x * 2.5
 9         print "Loop counter: ", j
10
11 print "Value of sum: ", sum
12 print "Value of y: ", y
```

```
$ python test6.py
Loop counter:  1
Loop counter:  2
Loop counter:  3
Loop counter:  4
Loop counter:  5
Loop counter:  6
Loop counter:  7
Loop counter:  8
Loop counter:  9
Value of sum:  112.5
Value of y:  8.625
```

Struktur data

Struktur data

- List
- List of list
- Tuple
- Dictionary
- String

- https://github.com/dudung/py-jupyter-nb/tree/main/src/stepin/builtin/list
- https://github.com/dudung/py-jupyter-nb/tree/main/src/stepin/builtin/loop
- https://github.com/dudung/py-jupyter-nb/tree/main/src/stepin/builtin/operators

Diskusi

Diskusi

- Mari berdiskusi
- Ada pertanyaan?
  - Contoh penggunaan for dan while yang lebih disarankan?
  - Penggunaan import untuk minggu depan?

Terima kasih

-