Flow of execution

Flow of execution

The *flow of execution* is the order in which program statements are executed.

Unless specified otherwise, program execution always begins from the first line. Statements are executed one at a time, top down.

Flow of execution

A function call creates a detour in the flow of execution.

- When a function is called, instead of going to the next line, the flow jumps to the first line of the function.
- When it reaches the last line of the function, the flow returns to where it left off.

Flow of execution

```
def happy():
  print 'Happy birthday to you'

def sing(anyone):
  happy()
  happy()
  print 'Happy birthday, dear', anyone + '.'
  happy()

sing('Joe')
print
sing('Ann')
```


Flow of execution

The *flow of execution* is the order in which program statements are executed.

Python provides three constructs to control the flow of execution.

- Functions
- Conditionals
- Iteration

Conditionals

Conditional constructs allow a program to execute different sequences of instructions for different cases.

The most basic form of conditional constructs is the *if* statement:

```
if <condition>:
  <statements>
```

Conditionals

The most basic form of conditional constructs is the *if* statement:

```
if <condition>:
  <statements>
```

- `<condition>` is a *boolean* expression.
- `<statements>` can be any sequence of statements.

Boolean expressions?

An *expression* is a code fragment that has a value.

Example.

```
1823
```


Boolean expressions?

An *expression* is a code fragment that has a value.

Example.

`1823 + n` (assuming `n` has a value)

Boolean expressions?

An *expression* is a code fragment that has a value.

Example.

```
'Trnity College'
```


Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

```
0 < 1
```


Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

`0 < 1` (whose value is `True`).

Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

```
0 > 1
```


Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

`0 > 1` (whose value is `False`).

Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

`temp < 32` (assuming `temp` has a value).

Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

```
temp >= 60 and temp <= 75
```


Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

```
temp < 60 or temp > 75
```


Boolean expressions

A *boolean expression* is a code fragment that has a `True/False` value (i.e., its value is either `True` or `False`).

Example.

```
not (temp < 32)
```

Comparators

To compare two numbers, Python uses:

Mathematics	Python
=	==
≠	!=
<	<
>	>
≤	<=
≥	>=

Conditionals

The most basic form of conditional constructs is the *if* statement:

```
if <condition>:
  <statements>
```

This simply means that, if `<condition>` is true, execute `<statements>` (otherwise do nothing).

Conditionals

Example.

```
if temp < 32:
  print "It's freezing outside."
```

This simply means that, if `temp < 32`, print `"It's freezing outside."`

Conditionals

Another important form of conditional constructs is the *if-else* statement:

```
if <condition>:
  <statements1>
else:
  <statements2>
```

This means that, if `<condition>` is true, execute `<statements1>`, or else execute `<statements2>`.

Conditionals

Example.

```
if temp < 32:
  print "It's freezing outside."
else:
  print "It's comfortable outside."
```

This means that, if `temp < 32`, print "`It's freezing outside.`" Otherwise, print "`It's comfortable outside.`"

Conditionals

If-else statements can be nested:

```
if grade >= 90:
  print 'A'
else:
  if grade >= 80:
    print 'B'
  else:
    if grade >= 70:
    ...
```

Conditionals

Nested if-else statements can be simplified with `elif` (which is just an abbreviation of `else if`):

```
if grade >= 90:
  print 'A'
elif grade >= 80:
    print 'B'
elif grade >= 70:
    ...
```


Dangling else

What would this output?

```
x = 7
if x < 5:
  print 'A'
  if x < 3:
    print 'B'
else:
  print 'C'
```

Output: `C`

Dangling else

What would this output?

```
x = 7
if x < 5:
  print 'A'
  if x < 3:
    print 'B'
  else:
    print 'C'
```

Output: nothing

Matching `else` with `if`

In the *if-else* statement

```
if <condition>:
  <statements>
else:
  <statements>
```

`else` is always matched against `if` with the same alignment.

Aligning statements

In the *if-else* statement

```
if <condition>:
  <statements>
else:
  <statements>
```

Every line in `<statements>` must be aligned together under the same indentation.

Aligning statements

What would this output?

```
x = 3
if x < 5:
  print 'A',
  print 'B',
else:
  print 'C',
  print 'D',
```

Output: `A B`

Aligning statements

What would this output?

```
x = 3
if x < 5:
  print 'A',
  print 'B',
else:
  print 'C',
print 'D',
```

Output: `A B D`

Proper indentation

Compared to other languages (such as C/C++, Java), proper spacing and indentation are very important in Python.

Note. To make indentation, always use the space key, not the tab key.

Long statement

If a single statement does not fit in a single line, do not just move to the next line.

```
print 'The current value is', val,
  'dollars.'
```

This will not work!

Long statement

If a single statement does not fit in a single line, do not just move to the next line.

```
print 'The current value is', val,\
  'dollars.'
```

The end-of-line character \ is required before starting a new line.

Multiple statements

If you need to put multiple statements in a single line, insert ; after each statement.

```
p = p * (1 + apr); print p
```