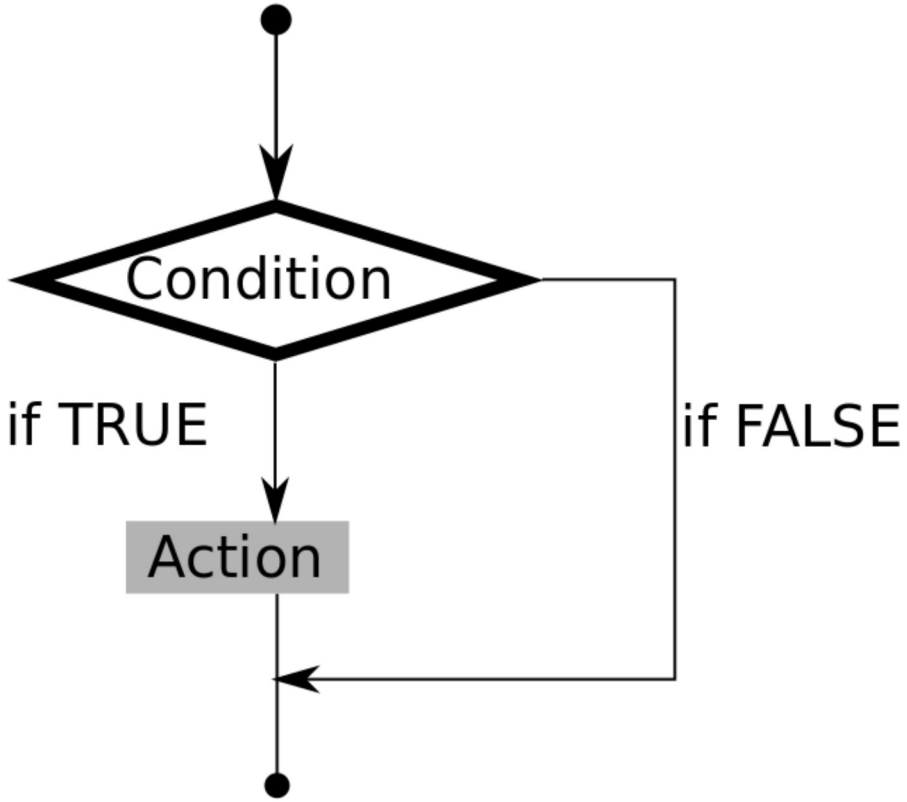
Table of contents (../toc.ipynb)

Conditions

- Decisions during execution of a program are at some stage almost everywhere needed.
- Conditions tell the computer: *do this if A happens, and do B if something else happens.*
- This gives programmer control about program execution.

Control flow



- These conditional statements are supported:
    - `if`
    - `elif` (the Python replacement for else if in other languages)
    - `else`

In [1]:

```
# Python treats zero as False
condition = 0

if condition:
    print("Catched")
else:
    print("Not catched")
```

```
Not catched
```

In [2]:

```
# Bool work of course perfectly

condition = True

if condition:
    print("Catched")
else:
    print("Not catched")
```

```
Catched
```

In [4]:

```
# Any arithmetic or comparision operator as well

signal = 5.5

if signal > 2.5:
    print("Signal overshoot")
```

```
Signal overshoot
```

Exercise: Conditions (10 minutes)

- Write a python function which checks if a number is positive, negative, or zero.
- The script should print the state (pos, neg, or zero) and the number.

Hint

- The function syntax (we will cover them later) in Python is

```
def my_func(args):
    your code goes here
```

- You might need `elif` for it.
- The print command might be `print("Positive number", x)` and the like.

Solution

Please find one possible solution in `solution_condition.py` (solution_condition.py) file.

In [27]:
```
import sys
sys.path.append("01_basic-python")

from solution_condition import *

check_value(0)
```

```
Is zero 0
```

In [6]:
```
check_value(11)
```

```
Is positive 11
```

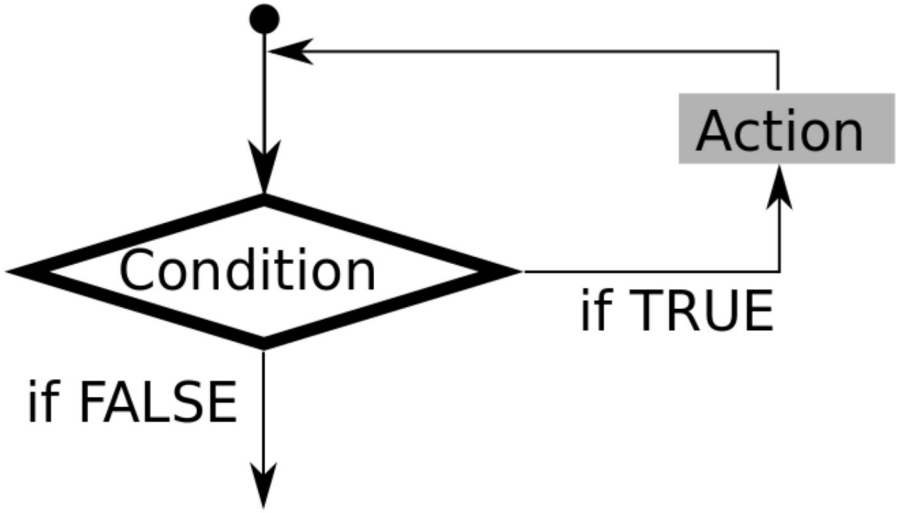
In [7]:
```
check_value(-3.14)
```

```
Is negative -3.14
```

Loops

- Loops are required to execute code multiple times.
- The basic loop keywords are
    - `while`
    - `for`
- Finer control inside loops is provided by
    - `break`
    - `continue`
    - `pass` statements

Control flow



Iterators

- Some common code snippets to define loop iterators are
    - `in range(start, stop)`
    - `in`
    - `in enumerate` which enumertates many objects like lists and returns index and item

For loop

- A for loop repeats a statement over a sequence.

In [1]:
```
for i in [0, 2, 3]:
    print(i)
```

```
0
2
3
```

In [2]:
```
for i in range(2, 5):
    print(i)
```

```
2
3
4
```

In [5]:
```
my_list = [1, 5, 22]

for idx, value in enumerate(my_list):
    print("Index=", idx, "Value=", value)
```

```
Index= 0 Value= 1
Index= 1 Value= 5
Index= 2 Value= 22
```

In [7]:

```
# Here a for loop with a break

for i in range(0, 99):
    print(i)
    if i > 5:
        print("Here the break")
        break
```

```
0
1
2
3
4
5
6
Here the break
```

In [4]:

```
# You can also loop over dictionaries

my_dict = {"power": 3.5, "speed": 120.3, "temperature": 23}

for field in my_dict.keys():
    print(field, "is adjusted to", my_dict[field])
```

```
power is adjusted to 3.5
temperature is adjusted to 23
speed is adjusted to 120.3
```

In [5]:
```
# Also strings work well

my_string = "Hello World"

for letter in my_string:
    print(letter)
```

```
H
e
l
l
o

W
o
r
l
d
```

For loop over two lists

- You can use `zip` to loop over multiple lists.

In [1]:
```
list_one = [0, 3, 5]
list_two = [8, 7, -3]

for i, j in zip(list_one, list_two):
    print(i * j)
```

```
0
21
-15
```

While loop

- Repeats a statement as long as condition is `True`.
- `while` loops are used if you do not know how long the sequence should be repeated.
- Condition is checked before code execution.
- You have to make sure that your while loops do not continue to infinity.
- `while` loops are barely used compared with `for` loops.

In [2]:
```
i = 0

while i < 4:
    i += 1  # do not forget to increment
    print(i)
```

```
1
2
3
4
```

In [6]:

```
"""Here a example for a while loop.
This loop will require different number of runs
until condition becomes True."""

import random
criterion = 9.5
sum_of_numbers = 0.0
idx = 0


while sum_of_numbers < criterion:
    sum_of_numbers += random.random()
    idx += 1

print(idx)
```

```
18
```

Exercise: For loop (5 minutes)

- Write a `for` loop which iterates over the values in dictionary

  ```
  {"force": [0, 10, 15, 30, 45], "distance": [2.5, 3.5, 6.0, -3.0, 8.1]}
  ```

  and computes the product of the two fields.

- Print the overall sum of these products.

Hint

- Think about if conversion of the dictionary makes sense.

Solution

Please find one possible solution in `solution_loop.py` (solution_loop.py) file.

In [2]:
```
%run solution_loop
```

```
399.5
```