# Lecture 3:
# Control Structures in Python

Introduction to Python
efl Data Science Courses

Jens Lausen

# Table of Contents

# 1. Program structures in general

- **Program structures** can be divided into

    1. **Simple assignments** like **variable definitions**, **value assignments** or **output commands** (Lectures 1 & 2)

    2. **Control structures:** a construct which enables to control the sequence of a program and allows complex calculation (This lecture)

- The structure of a program can be depicted in a so-called **Nassi-Shneiderman-diagram** or **structogram** (Nassi & Shneiderman 1972)

# 1. Program structures in general

### Process blocks

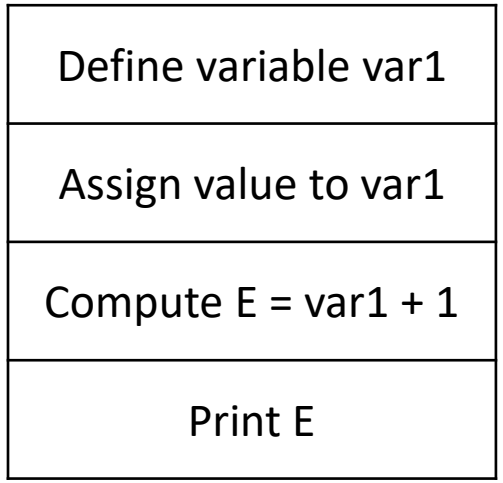
- No conditions
- Execute action inside of block and move to next block

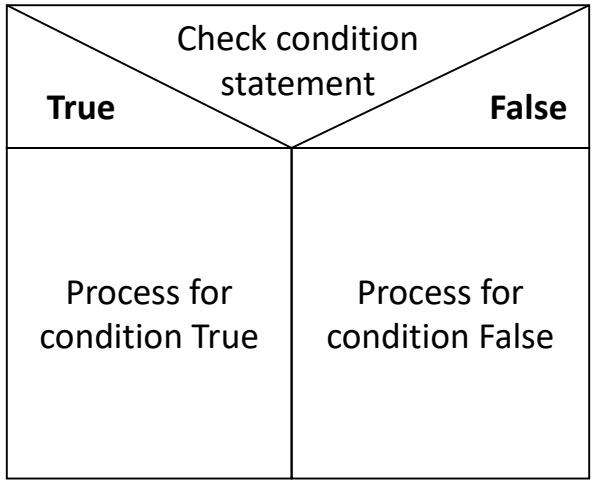
### Branching blocks

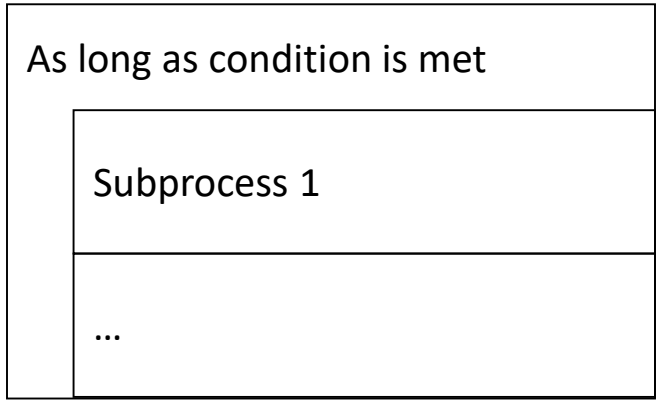
- Check condition statement
- Execute action inside of respective block

### Loops

- Loop a process as long as particular condition is met



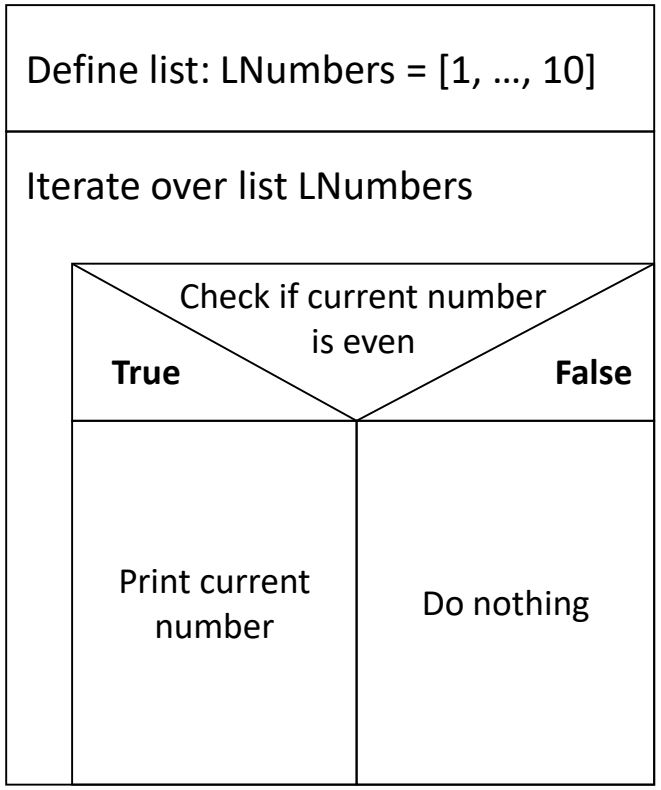



# Simple program example

Print all even numbers from a given list

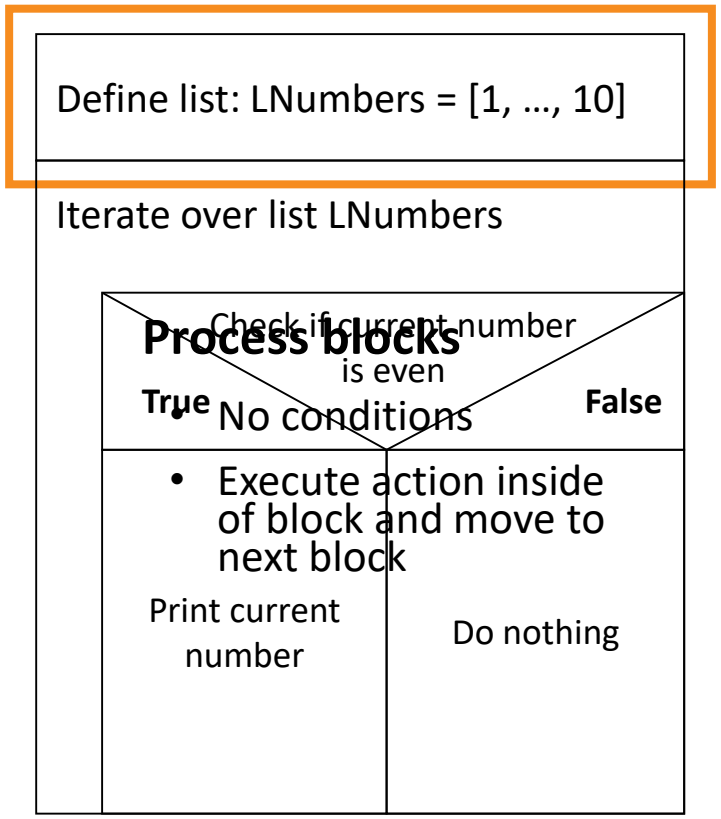
→ We will use this example for explaining the Python code in the following sections



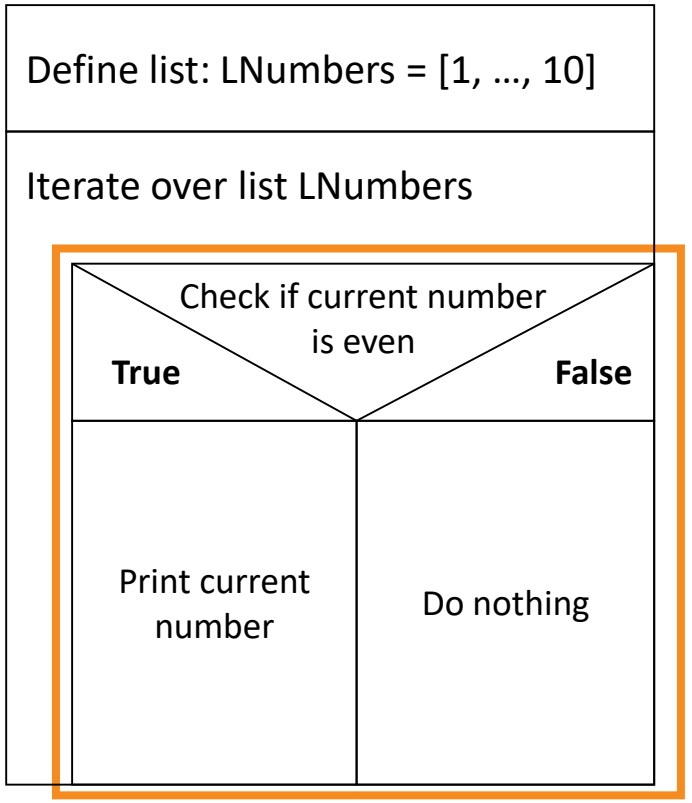
# Simple program example

Python code

```
# Define list of numbers from 1 to 10
LNumbers = [1,2,3,4,5,6,7,8,9,10]



# Alternative:
LNumbers = list(range(1,11))
```



# Simple program example



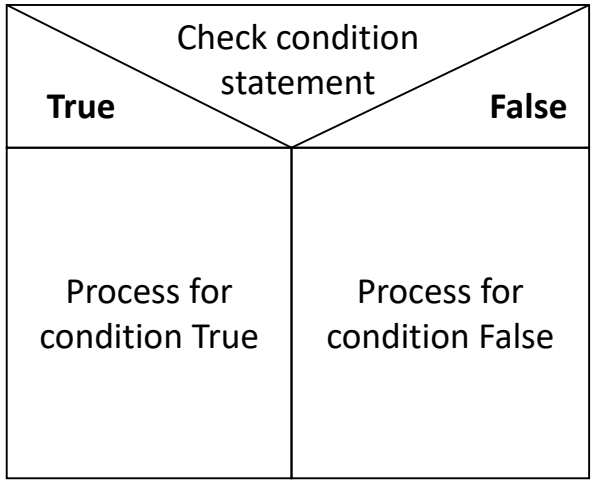
# Table of Contents

# 2. Case Distinction

In programming, for defining different conditional paths (or branches) for your code, so-called **if-statements** can be used.

- For each statement, the subordinate code will only be executed, if the underlying condition is fulfilled

- There are three types of if-statements:
  1. One-sided
  2. Two-sided
  3. if … elif … else ladder




**Branching blocks**

- Check condition statement
- Execute action inside of respective block

# 2. Case Distinction

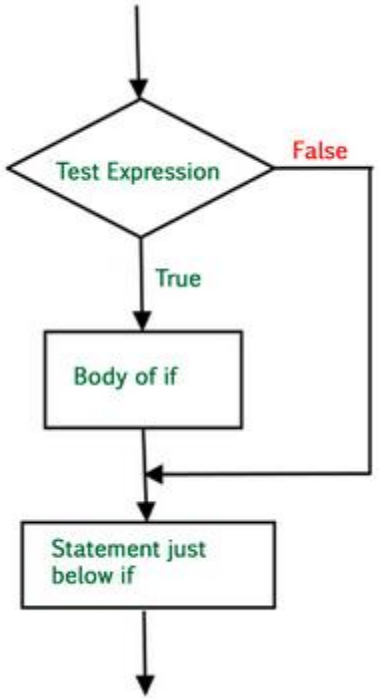
1. One-sided if-statement

**Python syntax**

```
if condition:
    statement
```

**Python example**

```
iNumber = 5
if iNumber < 10:
    print('Number is less than 10')
```




Source: https://www.geeksforgeeks.org/decision-making-python-else-nested-elif/

# 2. Case Distinction

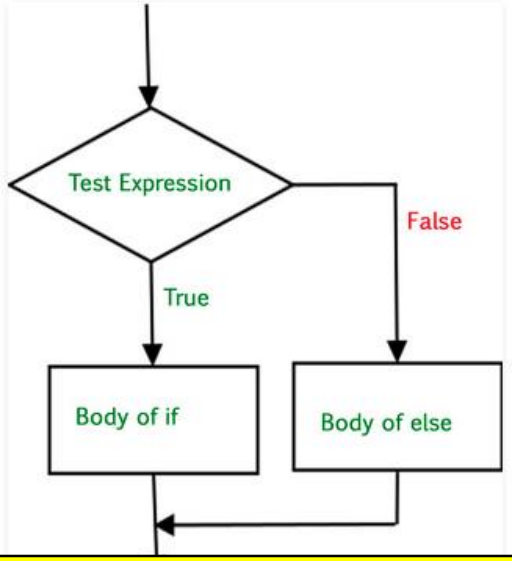
2. Two-sided if-statement

Python syntax

```
if condition:
    statement if condition True
else:
    statement if condition False
```

Python example

```
iNumber = 5
if iNumber < 10:
    print('Number is less than 10’)
else:
    print('Number is greater or equal to 10’)
```




Info:

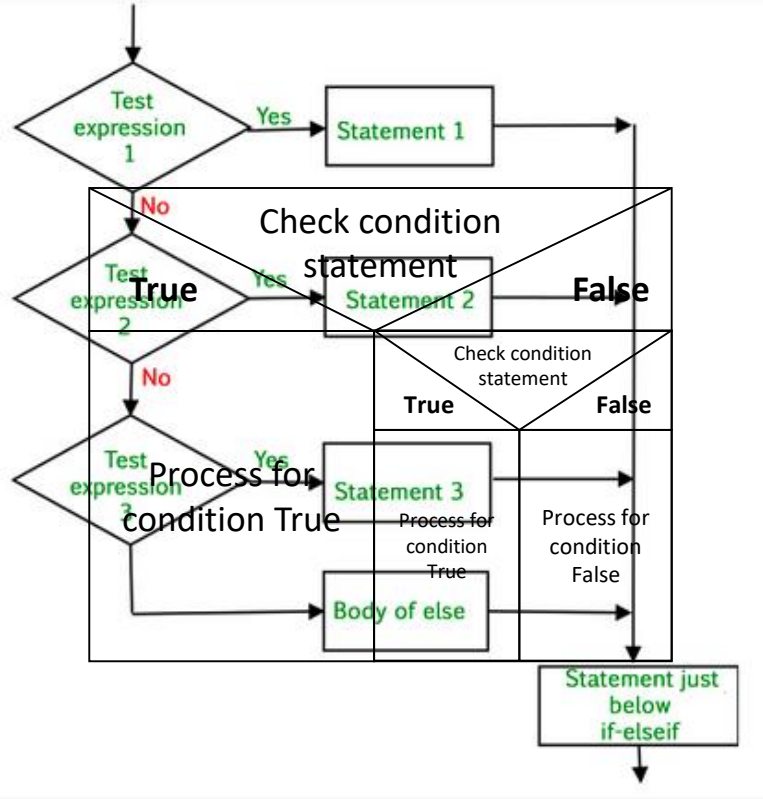
- The statement inside the body of an else-statement can be another if-statement
- Such program structures are called “nested if-statements”

# 2. Case Distinction

3. if … elif … else ladder

Python syntax

```
if condition:
    statement
elif:
    statement
.
.
else:
    statement
```




Source: https://www.geeksforgeeks.org/decision-making-python-else-nested-elif/

# 2. Case Distinction

- The condition in an if-statement is an expression with a boolean (`True, False`) as a results
- The conditional statement can also be a combination of multiple expressions
- For combining expressions, different operarators can be used

Python syntax

```
if condition:
    statement
else:
    statement
```

1. **Arithmetic operators**
   numeric → numeric

| | |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |

2. **Comparative operators**
   numeric → boolean

| | |
|---|---|
| < | Smaller |
| <= | Smaller or equal |
| > | Larger |
| >= | Larger or equal |
| == | Equal |
| != | Not equal |

3. **Logical operators**
   boolean → boolean

| | |
|---|---|
| and | If both operands are True then condition becomes True |
| or | If any of the two operands is True then condition becomes True |
| not | Used to reverse the logical state of ist operand |

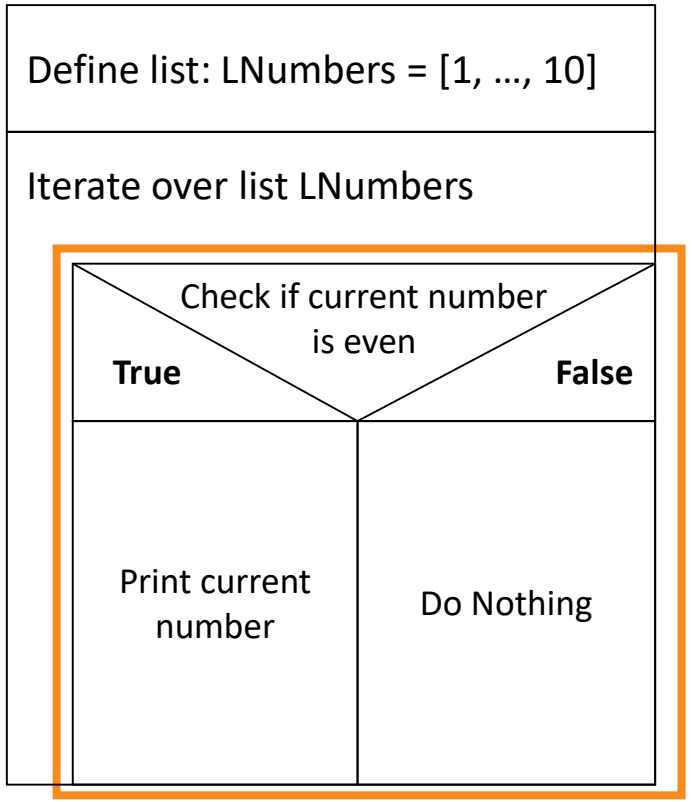
# Simple program example

Python code

```
# Define list of numbers from 1 to 10
LNumbers = [1,2,3,4,5,6,7,8,9,10]

iNum = LNumbers[0]

if iNum % 2 == 0:
    print(iNum)
```

Info:

- % is the modulo operator
- The modulo operation finds the remainder after division of one number by another
- Even numbers can always be divided by 2 with a remainder of 0



# Exercise 1: Case Distinction

efl the Data Science Institute

**1.1 Simple case distinction**

- Given are two random numbers
- Please write a conditional statement, which returns the minimum of both numbers

**1.2 Advanced case distinction**

- Please write a conditional statement, which returns the grade for an exam.
- Please use the following thresholds:
  - Points >= 90: Grade 1
  - Points >= 75: Grade 2
  - Points >= 60: Grade 3
  - Points >= 50: Grade 4
  - Points < 50: Grade 5

# Exercise 2: Booleans

**2. Conditional statements**

Given are the following expressions:

1. `(1+2) == 3`

2. `True != False`

3.
```
iNum = 4
(iNum != 0) and (20/iNum > 0)
```

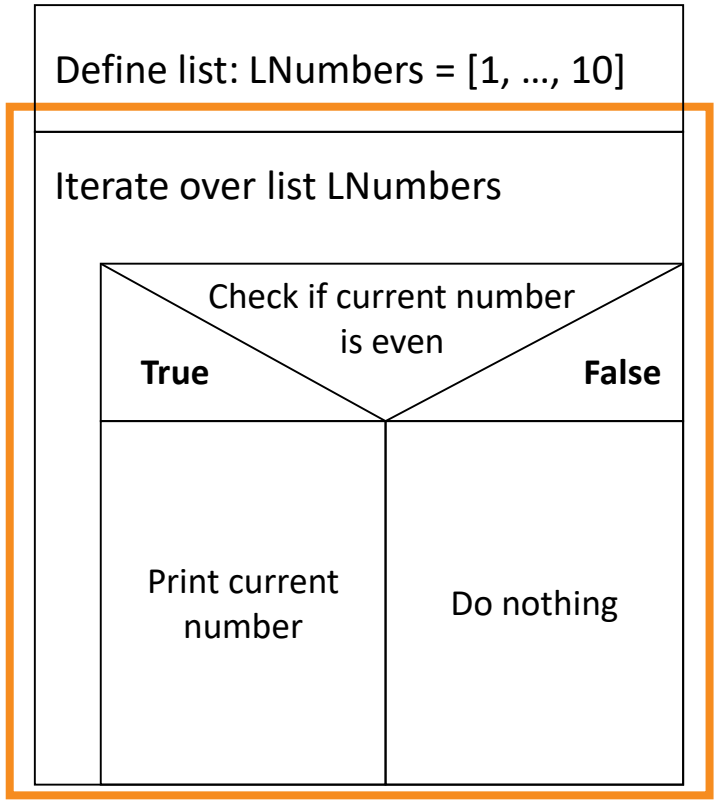
4.
```
iHour = 12
(iHour < 9) or (iHour > 18)
```

Please determine the result of each expression

# Simple program example

Python code

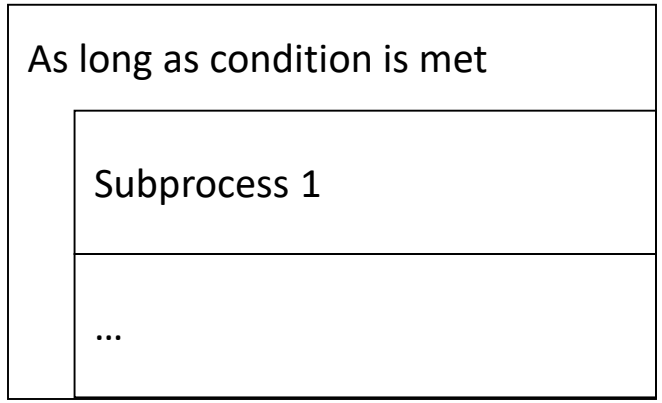
```
# Define list of numbers from 1 to 10
LNumbers = [1,2,3,4,5,6,7,8,9,10]

iNum = LNumbers[0]

if iNum % 2 == 0:
    print(iNum)
```



# Table of Contents

# 3. Loops

- Computers are often used to **automate repetitive tasks**
- A **repeated execution** of a set of statemens is called **iteration**
- Python has **two statements** for iteration:
    1. The **while** loop
    2. The **for** loop
- Further, both loop statements can be controlled by three different loop control statements, i.e.:
    1. continue
    2. break
    3. pass



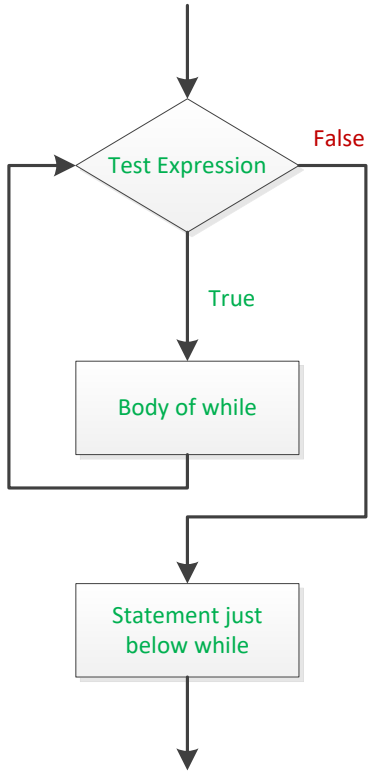

**Loops**

- Loop a process as long as particular condition is met

# 3. Loops

1. while loop:

- Executes a block of statements repeatedly until a given condition is satisfied
- When the condition becomse False, the statement immediately after the loop is executed




Python syntax

```
while condition:
    statement
```

Python example

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello!")
```

Output:

```
Hello!
Hello!
Hello!
```

# 3. Loops

## 2.1 for in loops:

- for loops are used for sequential traversal of iterable objects
- The sequence can be a list, tuple, string, dictionary, or any other iterable object

Python syntax

```
for iterator_var in sequence:
    statement
```

Python example

```
LNumbers = [1,2,3,4,5,6,7,8,9,10]

for iNum in LNumbers:
     print(iNum)
```

# 3. Loops

## 2.2 for loops using range

- the python build-in function **range** creates an iterable object which traverses all integer numbers in a pre-defined range
- All parameters forwarded to the range function need to be integers

Python syntax

```
range(stop)

range(start, stop)

range(start, stop, step)
```

- start = number to begin with
- stop = loop stops when this value is reached
  (the loop iterator variable will never reach this value)
- step = increment, by which the loop iterator variable increases in each iteration

Python example 1

```
# Iterate over range
for i in range(1, 10, 2):
    print(i)
```

Python example 2

```
# Iterating by index of sequences
LWords = ['Iterating', 'by', 'index', 'of', 'sequences']
for i in range(len(LWords)):
    print(LWords[i])
```

# 3. Loops

3. Loop control statements

- Change execution from its normal sequence

1. **`continue`**: returns the control to the beginning of the loop

```
for letter in 'science':
    if letter == 'e' or letter == 'c':
        continue
    print('Current Letter:', letter)
```

Output:
Current Letter: s
Current Letter: i
Current Letter: n

2. **`break`**: brings control out of the loop

```
for letter in 'science':
    if letter == 'e' or letter == 'c':
        break
    print('Current Letter:', letter)
```

Output:
Current Letter: s

3. **`pass`**: for empty loops

```
for letter in 'science':
    pass
print('Last Letter :', letter)
```

Output:
Last Letter: e

Examples based on: https://www.geeksforgeeks.org/loops-in-python/

# 3. Loops

Choosing between for and while:

- Use a **for** loop if you know, **before you start looping**, the **maximum number of times** that you'll need to **execute the body**
  - For example, if you're traversing a list of elements, you know that the maximum number of loop iterations you can possibly need is "all the elements in the list"

- By contrast, if you are required to **repeat some computation until some condition is met**, and you **cannot calculate in advance when this will happen**, you'll need a **while** loop

- We call the first case **definite iteration** — we have some definite bounds for what is needed

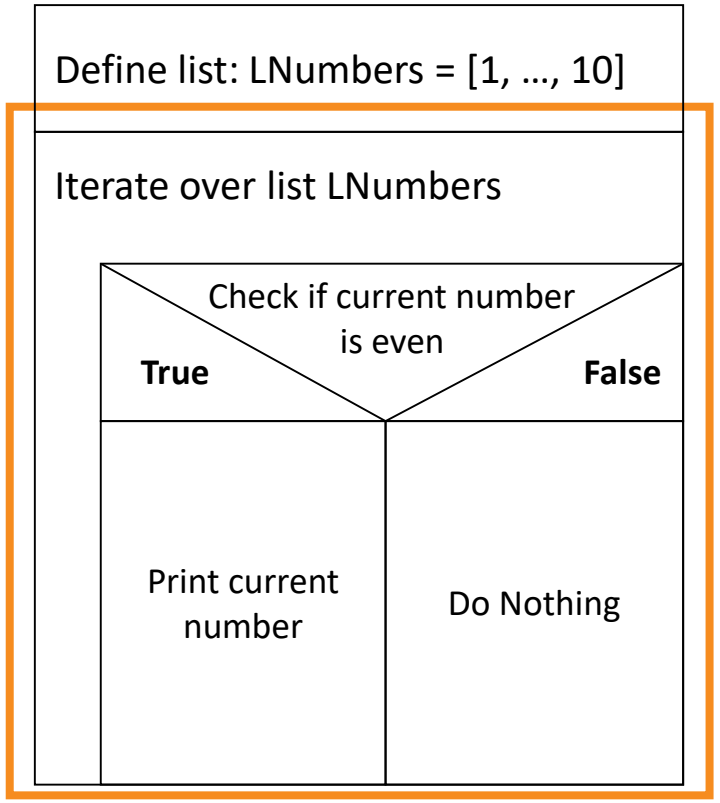
- The latter case is called **indefinite iteration** — we're not sure how many iterations we'll need

Based on: http://www.openbookproject.net/books/bpp4awd/ch04.html

# Simple program example

Python code

```
# Define list of numbers from 1 to 10
LNumbers = [1,2,3,4,5,6,7,8,9,10]

# Iterate over list
for iNum in LNumbers:
    # Check remainder of iNum
    if iNum % 2 == 0:
        # Print iNum if even
        print(iNum)
```



# Exercise 3: Loops

Given is the following List of numbers:

```
LNumbers = [5,23,37,49,50,46,30,46,70]
```

Please write a code that counts the number of elements in the list that are greater than 30, and prints the result

1. Use a while loop

2. Use a for loop

# References

Ernesti, J.; Kaiser, P.: Python 3 – Das umfassende Handbuch. 4. 2015.

Schiedermeier, R.: Programmieren mit Java. 2. 2010.

Stahlknecht, P.; Hasenkamp, U.: Einführung in die Wirtschaftsinformatik. 11. 2005.

https://docs.python.org/3/tutorial/

http://www.openbookproject.net/books/bpp4awd/ch04.html

https://www.geeksforgeeks.org/decision-making-python-else-nested-elif/

https://www.geeksforgeeks.org/loops-in-python/