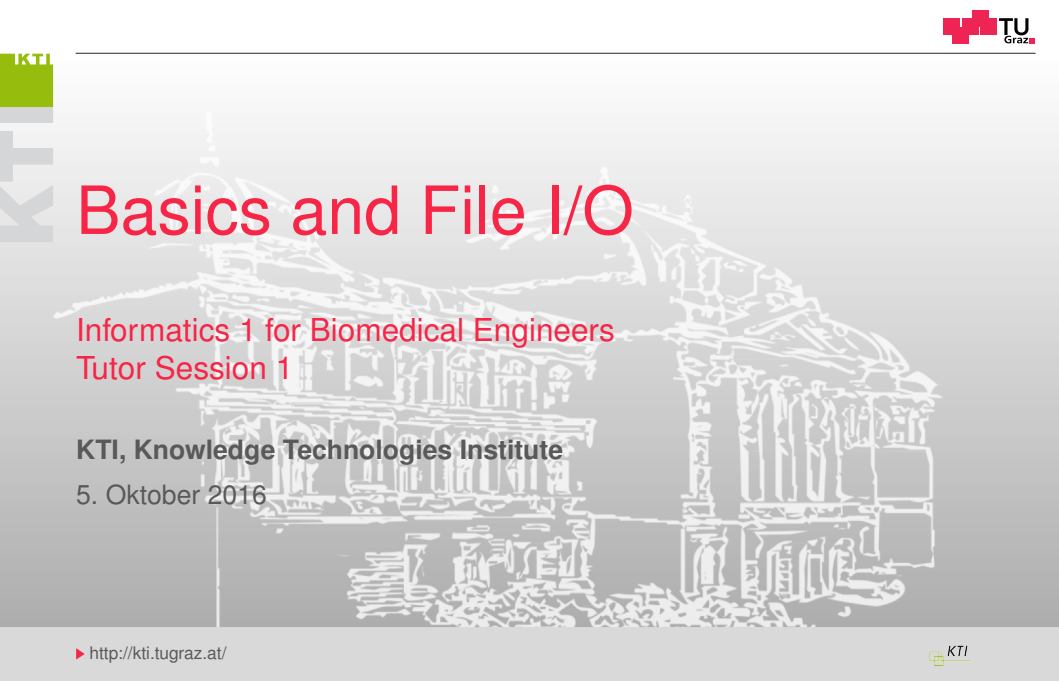
Basics and File I/O

Informatics 1 for Biomedical Engineers
Tutor Session 1

KTI, Knowledge Technologies Institute

5. Oktober 2016



Today's Topics

1. Indentation instead of braces – {}
2. Understanding the python documentation
3. Defining and using variables
   - Strings, Booleans, Integers, Floats
4. Using built-in functions
   - print(), range(), ...
5. Loops in python
6. Reading and writing files

Student Goals

- Be able to find reliable information on-line
- Know how to work with basic variables
- Write short programs solving simple numerical problems

Indentation - Scopes in Python

- Braces have a different meaning in python ... later on!
- Indents somewhat force readable code

```
1  #include <iostream.h>
2
3  int main()
4  {
5    std::cout << "Hello\n";
6    return 0;
7  }
```

```
1  if __name__ == "__main__":
2    print("World!")
```

Python documentation

- Use the official documentation [1]
- Recommended readings:
  - Python Tutorial [2]
  - Built-in Functions [3]
- Work of reference:
  - Python Standard Library Reference [4]

1 https://docs.python.org/3/
2 https://docs.python.org/3/tutorial/
3 https://docs.python.org/3/library/functions.html
4 https://docs.python.org/3/library/

Built-in help() Function

```
1   >>> help(abs)
2   Help on built-in function abs in module builtins:
3
4   abs(x, /)
5       Return the absolute value of the argument.
```

The print() statement

```
1  print('Only one argument given.')
2  #output: Only one argument given.
3
4  print('Two', 'Arguments, separated by whitespace')
5  #output: Two Arguments, separated by whitespace
```

Data types and variables

- Declaration by assigning an initial value
- Dynamically typed language
  - a variable is simply a value bound to a name
  - the value has a type, but the variable itself doesn't

```
1  i = 42
2  print('Value␣of␣i:␣', i)
3  #output: Value of i: 42
4
5  x = i * 2 + 1
6  print(x)
7  #output: 85
```

Data types and variables

- Using a variable on both sides (e.g. increasing it):

```
1  i = 42
2  i = i + 1
3  #new value of i: 43
4  #shorter:
5  i += 1
```

Data types and variables

- Using a variable on both sides (e.g. increasing it):

```
1  i = 42
2  i = i + 1
3  #new value of i: 43
4  #shorter:
5  i += 1
```

- Note: There's no i++ in Python

Data types and variables

Standard Data Types

- Integer

```
int
i = 42
```

Data types and variables

Standard Data Types

- Integer
- Floating Point Number

float

f = 42.0

Hint

Be careful with floats:
1.1 + 1.1 + 1.1 is not exactly 3.3

Data types and variables

Standard Data Types

- Integer
- Floating Point Number
- String

str
s = 'My string'

Data types and variables

Standard Data Types

- Integer
- Floating Point Number
- String
- Boolean

bool

b = True
or
t = False

Data types and variables

Data Type Conversion: *int()*, *float()* and *str()*

```
1  var = 42
2  var = str(var)
3  #var: '42'
4  var *= 2
5  #var: '4242'
6  var = float(var) + 0.5
7  #var: 4242.5
8  var = int(var)
9  #var: 4242
```

```
1  b = bool(1)
2  #b: True
3  b = not b
4  b = str(b)
5  #b: 'False'
6  #type(b) --> <class 'str'>
7
8  var = int(b)
9  #ValueError!
```

Operators

Symbols that represent operations

- Working with numbers
  - simple arithmetic operators:
    - `+, -, /, *`

```
1   #multiplication and division
2   # operators have precedence
3   # over the addition and
4   # subtraction operators:
5   3 + 1 * 5 - 1
6   #result: 7
7
8   (3 + 1) * (5 - 1)
9   #result: 16
10
11  3 / 2
12  #result: 1.5 (automatic casting)
```

Operators

Symbols that represent operations

- Working with numbers
  - simple arithmetic operators
  - modulo operator:
    - `modulo: %`

```
1  #modulo finds the remainder:
2  5 % 2
3  #result: 1
4  7 % 4
5  #result: 3
```

Operators

Symbols that represent operations

- Working with numbers
  - simple arithmetic operators
  - modulo operator
  - power operator:
    - **

```
1  #5 to the power of 2:
2  5**2
3  #result: 25
4
5
6  #this little trick
7  #gives us the square root:
8  9 ** (1/2)
9  #result: 3.0
```

Operators

Symbols that represent operations

- Working with numbers
- Working with strings
  - joining

    `str1 + str2`
  - multiply

    `str * number`

```
1   str1 = 'Hello'
2   str2 = 'World'
3   res = str1 + str(0) + str2
4   #res: 'Hello0World'
5
6   print('Hello', 'World')
7   #outputs the same as:
8   print('Hello␣' + 'World')
9
10  str = 'Hello' * 2
11  #str: 'HelloHello'
```

Operators

Symbols that represent operations

- Working with numbers
- Working with strings
- Boolean operators
  - `==` , `!=`
  - `>` , `<`
  - `>=` , `<=`
  - `and`, `or`, `in`, `not`, `is`

```
1   42 > 40
2   #True
3
4   42 >= 42 and 3 == 1 + 2
5   #True
6
7   (True and False) or (not 0)
8   #True
9
10  42 != 42
11  #False
12
13  1.1 + 1.1 + 1.1 == 3.3
14  #False
```

Python's Built-In Functions

Functions we already know:

- print() , abs()
- type()
- float(), int(), str()

Python's Built-In Functions

Functions we already know:

- print() , abs()
- type()
- float(), int(), str()

Other useful functions:

- input()
- round()
- range() ...

Retrieving User Input: input()

`input([prompt])`

The function then reads a line from input, converts it to a string and returns that.

```
1  years_str = input('How old are you: ')
2  days = int(years_str) * 365
3  print('Cool, ' + years_str + ' years are ' + str(days) + ' days.')
```

Branching

The if-Statement

```
1  if expression_1:
2      #instruction 1
3  elif expression_2: #optional
4      #instruction 2
5  else: #optional
6     #else instruction
```

Branching

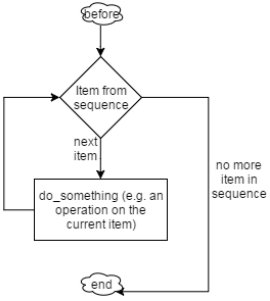
The if-Statement

```
1  weather = input('How is the weather today (rainy/sunny): ')
2
3  if weather == 'rainy':
4      print('clean your room!')
5  elif weather == 'sunny':
6      print('you can go swimming :-)')
7  else:
8      print('pff. don\'t have a recommendation.')
9  print('anyhow, watch a movie at night.')
```

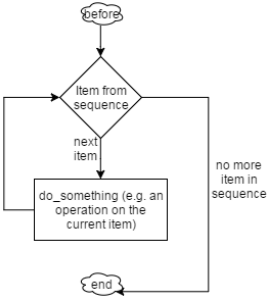
Online-Demo

For Loop

```
1  for item in sequence:
2      statement(s)
```



For Loop

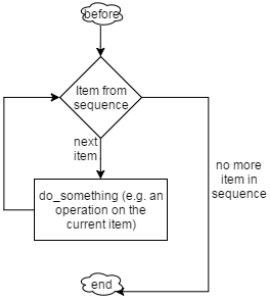



```
1  for item in sequence:
2     statement(s)
```

- No condition / stop criterion (like in other languages)

```
1  for ( init; condition; increment ) {
2     statement(s);
3  }
```

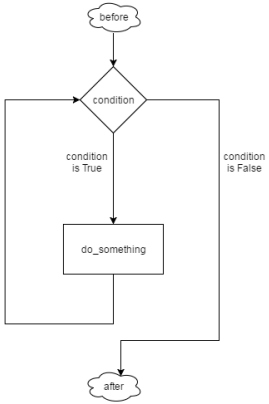
For Loop




```
1  for item in sequence:
2      statement(s)
```

```
1  #summarize even numbers
2  res = 0
3  for number in range(11):
4      if number % 2 == 0:
5          res += number
6  print(res)
```

While Loop




```
1  while condition:
2          #do_something
```

```
1  SECURE_PWD = '123'
2  user_inp = ''
3  attempts = 0
4
5  while user_inp != SECURE_PWD:
6          user_inp = input('Enter pwd: ')
7          attempts += 1
8
9  print('Authenticated after ', attempts,
10       'attempts')
```

File I/O - Using open()

- Python's built-in open() function opens a file and returns a *File Object*
- A file can be opened in several modes:
  - **r** - Reading a file
  - **w** - Open for writing (truncating the file first)
  - **a** - Open for writing (appending to the end if exists)
  - **x** - Create a new file and open it for writing
  - **b** - Binary mode (default is text mode)

File I/O - Reading files

- Reading the whole file

```
1  tf = open('textfile.txt', 'r')
2  content = tf.read()
3  tf.close()
```

- Reading line by line

```
1  tf = open('textfile.txt', 'r')
2  for line in tf:
3    print(line)
4  tf.close()
```

File I/O - Writing files

- Writing to a file

```
1  tf = open('textfile.txt', 'w')
2  for i in range(10):
3          line = 'Line number ' + str(i+1) + '\n'
4          tf.write(line)
5  tf.close()
```

- Writing to a binary file

```
1  bf = open('datafile.dat', 'wb')
2  bf.write( some-bytes.. )
3  bf.close()
```

Calculating π using Leibniz's formula

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}$$

written as a series:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots = \frac{\pi}{4}$$

Calculating π using Leibniz's formula

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}$$

```
1  ITERATIONS = 1000
2  subtotal = 0.0
3  for n in range(ITERATIONS):
4      subtotal += (-1)**n / (2*n + 1)
5  pi = subtotal * 4
6  print('Pi is appr.:', pi)
```

Task: Printing out the series

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots = \frac{\pi}{4}$$

- Write a program that prints the first six elements of the Leibniz series to the console (see right).

```
1 / 1
  -
1 / 3
  +
1 / 5
  -
1 / 7
  +
1 / 9
  -
1 / 11
  +
 ...
= pi / 4
```

Task: Printing out the series

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots = \frac{\pi}{4}$$

- Write a program that prints the first six elements of the Leibniz series to the console (see right).
- Hint: Use a While Loop (or For Loop) which runs from 1 to 11

```
1 / 1
  -
1 / 3
  +
1 / 5
  -
1 / 7
  +
1 / 9
  -
1 / 11
  +
 ...
= pi / 4
```

Task: Printing out the series

```
1   #possible solution:
2   last_denominator = 11
3   operator = '-'
4   act = 1
5   while act <= last_denominator:
6           print(1, '/', act)
7           print('␣', operator)
8           act += 2
9           if operator == '-':
10                  operator = '+'
11          else:
12                  operator = '-'
13  print('␣...\n=␣pi␣/␣4')
```