Session 4

1DV501 Python course

Loops

Today

- Iterations `while` and `for`
- Built-in and library functions
- A few built-in functions
- The `math` module
- The `random` module

Reading instructions: Sections 5.1-5.5, 6.1-6.4, 6.6

While

- In Python: `while` and `for` -statements
- Example: `while`

Reading instructions: Sections 5.1-5.5, 6.1-6.8

In [ ]:

```
print('1')
print('2')
print('3')
print('4')
```

In [ ]:

```
# Print 1 to 10 using while
n = 1

while n <= 10:
    print(n, end=' ')  # No line-break
    n += 1
print()    # Add line break
```

In [ ]:

```
# Print 1 to 10 using for

for i in range(1,11):
    print(i, end=' ')  # No line-break
print() # Add line break
```

In [ ]:

```
a = ['ett', 'två', 'tre']

for bokstav in a:
    print(bokstav)
```

The `while` statement

```
while "Test Condition":
   "Statements"
```

- The code "Statements" will be executed as long as "Test Expression"+ is `True`
- The code in "Statements" must be intended to be a part of the while statement
- "Statements" false execution jumps to the code after the while statement

 While-statement

In [ ]:

```
# Find smallest N such
# that 1+2+3+...+N > 100
s = 0   # sum
N = 0
while s <= 100:
    N = N + 1  # N = 1,2,3,4, ...
    s = s + N  # s = 1,3,6,10, ...
print("Smallest N is", N)
```

- Repeat certain statements (the loop body) as long as a condition is false
- The 2nd example (Smallest N) shows when to use a `while` statement, when we don't know how many iterations that are needed but we know when to stop.
- The 1st example is better handled by a `for` statement (coming soon) since we know exactly how many iterations that are needed (10).

Nestled examples

- Nestled statements within statement

In [ ]:

```
# Numbers dividable by 7 in range 1 to 100
n = 1

while n <= 100:
    if n % 7 == 0:  # Dividable by 7?
        print(n, end=' ')  # 7 14 21 ...
    n += 1
print()    # Add final line-break
```

In [ ]:

```
for n in range(0,101):
    if n % 7 == 0: print(n)
```

In [ ]:

```
# Do something while input is yes (y or Y)

entry = 'y'

while entry != 'N' and entry != 'n':
    entry = input("Enter Y to continue or N to quit: ")
    if entry == 'Y' or entry == 'y':
        print("Hello")    # Do something!
    elif entry != 'N' and entry != 'n':
        print(entry, "is not a valid input")
print("Done!")
```

In [ ]:

```
s = 'y'

print(s.upper())
```

Infinite loops

```
while True:
    print("Hello")
```

- `while True` : loop never stops
- Q: What happens when executed?
- A: It just runs and runs ... (You stop it by Ctrl-C in the Terminal window.)

In [ ]:

```
n = 1
str_ = ""    # empty string
while n < 10 or n > 0:
    n = n + 1
    str_ = str_ + "Hello"
```

- A logical error
- `n < 10 or n > 0+` is `True` for any `n`
- Program will crash since the string will get larger and larger and we will eventually run out of memory.

Infinite loops are often (but not always) a result of a logical error. They are sometimes useful when you want to do something (e.g. measure the temperature) without ever stopping. They do not harm your computer in any way.

The `for` Statement

In [ ]:

```
# Print 0,2,4,6,8,10
for i in range(0,11,2):
    print(i, end=' ')
print()
```

In [ ]:

```
# Countdown from 10
for i in range(10,0,-2):
    print(i, end=' ')
print()
```

In [ ]:

```
for i in range(100):
    print(i)
```

- `for i in range(0,11,2)` for each integer `i` in the range 0 to 10 using step size 2.
- **Notice:** The upper limit 11 is not included whereas the lower limit is.
- The variable `i` is called the *for counter*

The `range` function

The `range` function generates integer sequences and is rather powerful.

It comes in three versions:

- `range(stop)` : Considers by default the starting point as zero
- `range(start, stop)` : From `start` to `stop` with step size 1
- `range(start, stop, step)` : From `start` to `stop` with `step` size step
- **Notice**: Rather straight forward except that thestopvalue is not included.

In [ ]:

```
for i in range(10): print(i,end=',')
```

In [ ]:

```
for i in range(1,10): print(i,end=',')
```

In [ ]:

```
for i in range(1,10,2): print(i,end=',')
```

In [ ]:

```
for i in range(2,20,3): print(i,end=',')
```

In [ ]:

```
for i in range(14,-5,-2): print(i,end=',')
```

The keywords `break` and `continue`

- `break` and `continue` are used to jump out of a loop at an arbitrary position.

```
while bool_expr:
    ...
    if bool_expr:
      break  # End the loop, jump to next_statement

    if bool_expr:
      continue # End this iteration, jump to while bool_expr:
    ...
```

- `break` and `continue` are considered to make the code more difficult to understand ⇒ **use them with care!**

In [ ]:

```
n = 1
while True:
    print(n)
    if n > 10: continue
    n += 1
```

Is it a prime number? (Part 1)

$N>1$ is a prime number. Not dividable by any number in range 2 to $N-1$

Problem: Write a program that checks if a given N is a prime number.

Basic solution idea

- Error message if $N<2$
- For each integer i in range 2 to $N-1$
  - N dividable by i N is not a prime, interrupt loop
- Not dividable by any i N is a prime

```
Example runs

Enter an integer larger than 1: 47
47 is a prime number

Enter an integer larger than 1: 49
49 is NOT a prime number. It is dividable by  7

Enter an integer larger than 1: -7
Please follow the instructions!
```

Prime part 2

In [3]:

```
# Check if input is a prime number
n = int( input("Enter an integer larger than 1: "))

if n < 2:    # Must be larger than 1
    print("Please follow the instructions!")
else:
    prime = True
    for i in range(2,n):  # Check if prime
        if n % i == 0:
            prime = False
            break         # Jump from loop when not prime

    if prime:             # Present result
        print(n, "is a prime number")
    else:
         print(n, "is NOT a prime number. It is dividable by ",i)
```

```
7 is a prime number
```

Nestled statements

In [6]:

```
# Print multiplication table
n = int( input("Please enter a positive integer: "))

if n < 1:
    print("Input must be positive!")
else:
    print("Multiplication table for ", n)
    for i in range(1,n+1):
        for j in range(1, n+1):
            print(i, " x ", j, " = ", i*j)
```

```
Multiplication table for  3
1  x  1  =  1
1  x  2  =  2
1  x  3  =  3
2  x  1  =  2
2  x  2  =  4
2  x  3  =  6
3  x  1  =  3
3  x  2  =  6
3  x  3  =  9
```

Problem solving with `if`, `while`, and `for`

- Understanding each control statement by itself is rather easy
- Solving problem requiring only one such statement is also often rather easy
- However, many problems require multiple nestled control statements
- Solution with nestled statements -> much harder -> much training needed
- We have a large number of such problems in Assignment 2

Example: Count A

Write a program `countA.py` that reads a string from the keyboard and then prints how many `a` and `A` the string contains. An example of what an execution might look like:

```
Provide text:

Bedevere: What makes you think she is a witch?
Peasant: Oh, she turned me into a newt!
[Bedevere gives him a disbelieving look]
Bedevere: A newt?
[Silence]
Peasant: Well, I got better.
Peasant Crowd: Burn her anyway!.

Number of 'a':  13
Number of 'A':  1
```

Sketch of a Solution

1. Read a line of `text` (a string text)
2. For each character `c` in `text`
    - if `c = 'A'` increase counter `nA` by 1
    - `else if c = 'a'` increase counter `na` by 1
3. Print result Print `+nA+ and +na+`

Hint: I should have waited with 1 (read line of text) until the end. Why?

In [10]:

```
s = 'This is a string'

for n in s:
    print(n,end='')
```

```
This is a string
```

In [8]:

```
# Count number of 'A' and 'a' in a string
text = input("Please provide text: ")

na, nA = 0, 0

for c in text:
    if c == 'a':
        na += 1
    elif c == 'A':
        nA += 1
print("\nNumber of 'a': ", na)
print("Number of 'A': ", nA)
```

```
Number of 'a':  2
Number of 'A':  2
```

Iterating over all characters in a string is simple using a for statement

```
text = "Peasant Woman: Didn't know we had a king. I thought we were an autonomous c
ollective."
for c in text:
    "Do something with character c"
```

break

Using library functions

In [11]:

```
import random



# Print random numbers
n = int ( input("Number of random numers: "))

print(n, "random numbers: ", end=" ")
for i in range(n):       # n iterations
    r = random.randint(1,1000)
    print(r, end=" ")
print()
```

```
4 random numbers:  327 763 596 300
```

In [14]:

```
import random
random.getrandbits(4)
```

Out[14]:

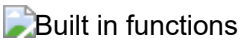
```
6
```

- The program above uses 5 different functions
- Built-in functions: `input()`, `int()`, `print()`, `range()`. These functions are always available
- Library functions: `randint()`
- `randint` belongs to the module `random`

- We will present a number of functions that might me useful in your assignments in the following slides.
- Both built-in functions (always available) and library functions (requires import).

Built in functions

- Built-in functions are always available ⇒ no import needed

 Built in functions

- About a third of the functions above will be presented and used in this course

Common built in functions

- `max(), min(), abs(), round()`
- `float(), int(), bool(), str()` type conversion
- `len("Hello")` length of something. In this case a string
- `type()` current type of variable or expression

In [15]:

```
a, b, c, = 1.0, -2.2, 3.14

print("Values:",a, b, c, sep=", ")   # 1.0, -2.2, 3.14
print("Maximum:", max(a,b,c))        # 3.14
print("Minimum:", min(a,b,c))        # -2.2
print("Absolute value:", abs(b))     # 2.2
print("Round off:", round(c))        # 3
print("Hello", len("Hello"))         # Hello 5
print(type(a), type("Hello"))        # <class 'float'> <class 'str'>
```

```
Values:, 1.0, -2.2, 3.14
Maximum: 3.14
Minimum: -2.2
Absolute value: 2.2
Round off: 3
Hello 5
<class 'float'> <class 'str'>
```

The math module

In [16]:

```
import math
pi = math.pi                    # Pi as a float

print(pi)                       # 3.141592653589793
print( math.degrees(pi/3) )     # 60
print( math.cos(pi/3) )         # 0.5
print( math.modf(pi))           # Return the fractional and integer parts of x

print( math.sqrt(2) )            # 1.4142135623730951
print( math.pow(4,3) )           # 64.0
print( math.floor(pi), math.ceil(pi) )         # 3 4
print( math.gcd(15,20) )         # 5  the largest positive integer that divides each of the
integers


# https://docs.python.org/3/library/math.html
```

```
3.141592653589793
59.99999999999999
0.5000000000000001
(0.14159265358979312, 3.0)
1.4142135623730951
64.0
3 4
5
```

- `math` is an external module \ra must be imported
  - Trigonometric functions: *sin(), cos(), tan(), asin(), acos(), atan()*, ...They work with radians by default
- `math.degrees( pi/3)` convert radians to degrees
- `sqrt(x), pow(x,p)` square root of x, x raised to the power of p
- `floor(x), ceil(x)` Rounds off x downwards/upwards to the nearest integer
- `gcd(n,p)` Greatest common devisor

Import modules and functions

- Three equivalent import approaches

```
#1st approach

import math                # Make math module available

print( math.sqrt(2) )        # Must make reference to math module
print( math.gcd(4,3) )
```

```
# 2nd approach

from math import sqrt, gcd  # Make sqrt and gcd from math available

print( sqrt(2) )             # No math reference required
print( gcd(10,15) )
```

```
# 3rd approach (not recommended)

from math import *         # Make all functions in math available

print( sqrt(2) )            # No math reference required
print( gcd(10,15) )
```

Note:

- Use 1st approach when multiple math functions are needed.
- Use 2nd approach when only 1-2 functions are needed
- **Avoid 3rd approach since name collisions are more likely**

Short names for imported modules

from math

In [18]:

```
from math import sqrt as kvadratroten
```

In [ ]:

```

```

In [20]:

```
import random                    # Make random module available

print( random.randint(0,100) )   # Random int in [0,100]
print( random.uniform(20,30) )   # Random float in [20,30]
```

```
33
20.92231572959579
```

In [21]:

```
import random as rd              # Give random the name "rd"

print( rd.randint(0,100) )       # Use "rd" to reference random
print( rd.uniform(20,30) )
```

```
96
24.649620303216835
```

- Use 1st approach when module name is short (e.g. `math`)
- Use 2nd approach when module name is lengthy (e.g. `matplotlib`)

Common short names

- `numpy as np`
- `pandas as pd`
- `matplotlib.pyplot as plt`

In [22]:

```
# Example

from random import randint as slumpvis_siffra

print(slumpvis_siffra(0,100) )
```

```
80
```

Time

- We use function \verb+time()+ in module \verb+time+ to measure time
- `time.time()` gives the time in seconds since epoch (January 1, 1970)
- `start = time()` gives a starting point and ...
- `elapsed = time() - start` gives elapsed time since `start`

In [23]:

```
# Measure time
from time import time

start = time()        # start timer
name = input("Enter you name: ")
elapsed = time() - start   # Measure time difference since start
elapsed = round(elapsed,1)
print("It took you", elapsed, "seconds to enter your name")
```

```
It took you 2.9 seconds to enter your name
```

In [27]:

```
import time
time.sleep(1)
```

In [ ]:

```

```

Terminate execution using `exit()`

- Handle initial check using `if-else`

```
n = int( input("Enter a positive integer: ") )

if n < 1:
    print("Input must be positive")
else:
    "Do something with n ... ==> main part of program"
```

- Handle initial check using `if` and `sys.exit()`

```
import sys

n = int( input("Enter a positive integer: ") )

if n < 1:
    print("Input must be positive")
    sys.exit()   # Terminates program execution

"Do something with n ... ==> main part of program."
```

- **Advantage:** Avoid having to indent main part of program.
- **Disadvantage:** `exit()` terminates program prematurely ⇒ sometimes a bit harder to understand.

?segnälkaB

- .netxet åp adnäv nak mos margorp tte virkS

In [28]:

```
# First try

s= input('Reverse:')

for i in range(len(s)):
    out += s[len(s)- (i) ]

print(out)

del out
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-28-0247d467ba0d> in <module>
      4
      5 for i in range(len(s)):
----> 6     out += s[len(s)- (i) ]
      7
      8 print(out)

NameError: name 'out' is not defined
```

In [34]:

```
s= input('Reverse:')

out=''

for i in range(len(s)):
    out += s[len(s)-(i+1)]

print(out)

del out
```

```
jeh
```

In [32]:

```
len(s)
```

Out[32]:

```
3
```

In [ ]:

```
for i in range(26):
    print((i + 5) % 26)
```

Example: Simple Encryption

A very simple way to encrypt a text would be to just shift each letter one step in the alphabet. That is, replace all letters in the text with the next letter in the alphabet. **Caesar cipher**

 Cipher

```
 a --> b, b --> c,    ...    , y --> z, z --> a
A --> B, B --> C,    ...    , Y --> Z, Z --> A
```

- All non-letters, for example digits, ? ,!, \%, and whitespace, are left unchanged.

Exercise:

- Write a program `stupidencryption.py` that reads a line of text from the user and presents an encrypted version of the text according to the encryption method outlined above. An execution might look like this:

  ```
  Provide a line of text: Was it a rat I saw?
  Encrypted Text: Xbt ju b sbu J tbx?
  ```

- **Hints**: A-Z have ASCII codes in range [65,90], a-z are in range [97,122], built-in function `ord()` gives the ASCII code for a character, and function `chr()` gives the character for an ASCII.

- **Basic idea:** Convert each letter to ASCII, add 1, and convert back to character

Encrypted message as follows:

```
 J gbsu jo zpvs hfofsbm ejsfdujpo! Zpvs npuifs xbt b ibntufs boe zpvs gbuifs tnfmu pg
fmefscfssjft!
```

ASCII Table

In [38]:

```
ord('A')
```

Out[38]:

```
65
```

In [39]:

```
chr(65)
```

Out[39]:

```
'A'
```

In [40]:

```
# str = "abcdefghijklmnopqrstuvwxyz 123 ABCDEFGHIJKLMNOPQRSTUVWXYZ.;?"
str_ = input("Provide a line of text: ")

result = ""
for c in str_:
    asc = ord(c)   # ASCII code
    if c == 'z':   # z --> a
        result += 'a'
    elif c == 'Z': # Z --> A
        result += 'A'
    elif 65 <= asc <= 90:  # Upper case
        result += chr(asc+1)
    elif 97 <= asc <= 122: # Lower case
        result += chr(asc+1)
    else:                  # Handle non-characters
        result += c
print("Encrypted text:", result)
```

```
Encrypted text: Ifk
```

In [42]:

```
# str = "abcdefghijklmnopqrstuvwxyz 123 ABCDEFGHIJKLMNOPQRSTUVWXYZ.;?"
str = input("Decrypt: ")

result = ""
for c in str:
    asc = ord(c)   # ASCII code
    if c == 'a':   # z --> a
        result += 'z'
    elif c == 'A': # Z --> A
        result += 'Z'
    elif 65 <= asc <= 90:  # Upper case
        result += chr(asc-1)
    elif 97 <= asc <= 122: # Lower case
        result += chr(asc-1)
    else:                  # Handle non-characters
        result += c
print("Decrypted text:", result)
```

```
Decrypted text: I fart in your general direction! Your mother was a hamster a
nd your father smelt of elderberries!
```

Old Java-test Exercise

Write a Java program `square` that first reads any integer (higher than or equal to 3) from the keyboard and then prints a non-filled square of the type presented below.

An execution might look like this:

```
Provide an integer 3 or higher: 5
The square for number 5
  *****
  *   *
  *   *
  *   *
  *****
```

- An error message should be given, and the program should terminate, if the user-provided integer value is below three.

In [43]:

```
sz = int ( input("Enter an integer 3 or higher: "))

if sz < 3:  # Check input
    print("The size must 3 or higher")
else:
    # Two types of square lines
    stars = ""
    for i in range(sz):
        stars += "*"

    line = "*"
    for i in range(sz-2):
        line += " "
    line += "*"

    # Print square
    print("\nThe square for number", sz)
    print(stars)
    for i in range(sz-2):
        print(line)
    print(stars)
```

```
The square for number 5
*****
*   *
*   *
*   *
*****
```

In [ ]:

```
math.pi
```

In [ ]:

```
x = ""
```

In [ ]:

```
x += "ab"
```

In [ ]:

```
x
```

In [ ]:

```
math.modf(20.5)
```

In [ ]:

```
s = 'Tar vi detta baklänges'

out=''
for i in range(len(s)):
    out += s[len(s)-(i+1)]

print(out)
```

In [ ]:

```
len(s)
```

In [ ]:

```
out = s[4]
```

In [ ]:

```
out
```

In [ ]:

```

```