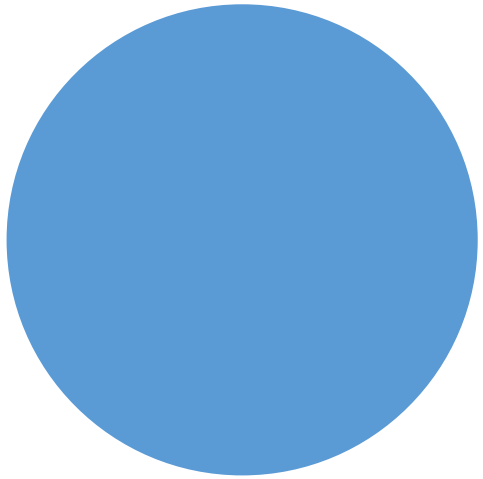
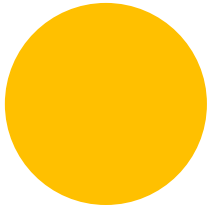

Programming I

Course 5

Introduction to programming

What we talked about?

- Functions
  - Function definition
  - Functions call
- Variables visibility domain
  - Local
  - global

What we will talk about?

- Modules
- Characters
- Strings
- Formatting Strings
- Regular expressions

Modules in Python

- What is a module?
  - A set of functions grouped into a file, which can be accessed
- Modules types (libraries)
  - Predefined (build-in)
  - External (created by you, created by somebody else)
- Example
  - math – mathematical functions (build-in)
  - Random – random number generator (build-in)
  - cv2 – image processing (external)
  - numpy – fast operations with arrays (external)

How to use

import

- Allows the import of all function from module
- Not recommended

```
import math
print(math.sqrt(100))
print(math.log10(100))
```

from

- Allows the import of a individual function from module
- Recommended

```
from math import sqrt
from math import log10
print(sqrt(100))
print(log10(100))
```

Rename Modules

- Using **as** keyword

import

- Allows the import of all function from module
- Not recommended

```
import math as m
print(m.sqrt(100))
print(m.log10(100))
```

from

- Allows the import of a individual function from module
- Recommended

```
from math import sqrt as sq, log10 as lg
print(sq(100))
print(lg(100))
```

What we will toke about?

- Modules
- **Characters**
- Strings
- Formatting Strings
- Regular expressions

Characters

- Character representation
  - a translation scheme is used which maps each character to its representative number
- Schems
  - ASCII
    - covers the common Latin characters
  - Unicode
    - provide a numeric code for every possible character, in every possible language, on every possible platform
- Functions
  - chr() - Converts an integer to a character
  - ord() - Converts a character to an integer

Example:

```
print(ord('a'))
print(chr(97))
```

Strings

- Arrays of characters
- Can be defined
  - Using " or '
- Example
  - 'a valid string', not ~~`an invalid string `~~
  - "a valid string"
  - "another' valid string"
  - 'another "valid string'
  - ~~"not a valid string'~~

Escape characters

- Charter with special meaning
- Formed from \ followed from a character
- Example
  - Use " inside a string specified using "
    - "a new \"cuvant\""
  - New line \n
    - "first line\n second line"
  - Tab \t
  - Backspace \b
- For more details see: https://docs.python.org/2.0/ref/strings.html

Strings

- An ordered characters sequence
  - Similar with list
  - Allow indexing
    - Accessing character ay position
    - Slicing
- Example

  S="hello

  S[0] -> h

  S[1] -> e

  S[::-1] -> olleh

  S[1:3] -> el

- Number of characters into a string
  - len()
- Example

  S="hello"

  print(len(s)) -> 5

Other String Methods

- +
  - Concatenates 2 string values
- Example

  S1="ab"

  S2="cd"

  S1+S2 -> "abcd"

- *
  - Multiply a string value
- Example

  S1="ab"

  S1*3 -> "abababab"

Strings. Immutable

- Immutable
  - in order to change the content of an object a new object is created
- Example

  S="course"

  S[0]="C" -> ERROR: 'str' object does not support item assignment

  Possible solution

  S = "C"+S[1:]

Other String Methods

- upper()/lower()
  - Transform a string to upper/lower cases
  - The functions are not destructive (do not change the content of the object)
- Example

  S = "Hello"

  S.upper() -> "HELLO"

- split([separator[, max]])
  - Returns a list with elements from the string that are separated by *separator* (default value is space) having *max* elements (default value for *max* is -1 meaning all elements)
- Example

  S="red green blue"

  s.split() -> ["red", "green', "blue"]

  S="red;green;blue"

  s.split(";") -> ["red", "green', "blue"]

Other String Methods

- join(sequence)
  - Return a string that contains the elements of the sequence separated by the string value on which the function is applied

- Example

  " ". join(["red", "green", "blue"]) -> "red green blue"

  " and ". join(["red", "green", "blue"]) -> "red and green and blue"

  " ".join([1,2,4]) -> ERROR expected sequence str

Exercise

- Write a program that draws the following triangle for input number *n.* For example for *n=4* the result is the following

```
*
**
***
****
```

Solutions?

1. Using * operator for strings
2. Using + operator for strings
3. Using formatted *print* function

Formatting Strings

- Remember: string are immutable
- How many string objects are creating in order to evaluate the following line?

```
S = " This " +  " course " + " is " + " about " +  " strings"
```

- What is result of evaluating the following sequence?

```
A=10
B=20
S="a=" + a + " b=" + b
```

Formatting Strings

- Method to insert values into a string
  - Preserve arguments order
  - Change arguments order
- Example

```
A=10
B=20
S="a={}  b={}".format(a, b)
S="int value={}  string value={}".format("abc", a)
S="a={1}  b={2}".format(a, b)
S="a={2}  b={1}".format(a, b)
```

Formatting Strings

- Numbers
  - Number of decimal digits

    F = 1.2334456

    #display first 2 decimals

    print("f={:1.2f}".format(F))

  - Padding numbers

    i=12

    #display number on 4 characters

    print("i={:4d}".format(i))

Formatting Strings

- Padding and alignment

  '{:>10}'.format('test') #align right

  '{:10}'.format('test') #align left

  '{:_<10}'.format('test') # use _ in steed of space

  '{:^10}'.format('test') # center

- Truncating long strings

  '{:.5}'.format('xylophone')

Formatting with 'f' or 'r'

- Use f-string in seed of format

```
name="Ana"
age="20"
print(f"Student {name} has {age}.")
```

- Use r- (raw) to disable escape sequence

```
print("ana\has\n apples\n") => ana
                                has
                                apples
print(r "ana\has\n apples\n") => ana\has\n apples\n
```

Usual String Operations

- Checking if a substring belongs to a string
  - Solution
    - Implement your own search algorithm
    - Use predefined functions from the language
      - Example
        - find() – returns the index of the fist appearebce of the substring in string

Usual String Operations

- Verify if a string is in correct format
  - A value is an e-mail address
  - A date is entered in the correct format
- Split a string by a delimiter
  - comma separated values information
  - Find the words from a sentence

Regular Expressions - Regex

- A compact notation for. String representation
- Defined using a mini-language
  - Depends on programming language
  - Independent of programming language
- Useful
  - Validating string format
  - Splitting strings after a criteria
  - Searching
  - Searching and replacing

Regex Characters

- Used to search an exact string
  - Example regex='abc'
    - aabcc
    - ababc
    - ~~aabbcc~~
- Escape characters
  - `\.^$?+*{}[]()|`
  - Have a significance if they are encountered inside a regex
  - Example: regex=ab+c #match multile b instances 'b+'
    - aabcc
    - ababc
    - aabbcc

Regex Characters

- Escape characters
  - `\.^$?+*{}[]()|`
  - Have a significance if they are encountered inside a regex
  - Example: regex=a[bc]d #match multile b instances 'b' or 'c'
    - abd
    - acd
    - ~~abcd~~
    - ~~acbd~~
    - aabdddd
    - acdd

Regex Characters

- Escape characters
  - `\.^$?+*{}[]()|`
  - Example
    - [0-9] matching a digit
    - [^0-9] matching any character that us not a digit

Regex Characters – shortcuts

Symbol	Semnification
.	any character except new line
\d	any digit
\D	any non-digit character
\s	white characters ([\t\n\r\f\v])
\S	any non white characters
\w	any “word” ([a-zA-Z0-9_])
\W	any non-word

Regex. Example

- Regex to identify/validate dates into a text
  - ***15/10/2018***
  - ***Regex="***`[0-9][0-9]/[0-9][0-9]/[0-9][0-9][0-9][0-9]`***"***
  - ***Regex="***`\d\d/\d\d/\d\d\d\d`***"***
  - ***Regex="\d{1,2}/\d{1,2}/\d{4,4}"***
- Regex to validate e-mail
  - Formed: any letter, digit and _, followed by @
    - ***Regex="***`\w@[a-z] `***"***
    - string: `marian`@google.com
    - result: n@g

Regex - Quantifiers

- Syntax: {min,max}
  - Say how many times the expression is repeating
  - Example
    - regex - a{1,1}a{1,1} =>aa
    - regex - a{1,2} >a or aa

- For an exact number of repetitions it is enough {number}
- If no specifiers, the default value is 1

Regex - Quantifiers

- {0,1} equivalent with ?
  - example: regex - ab{0,1}c
  - example: regex - ab?c

- {1,n} equivalent with +
  - At list one appearence

- {0,n} equivalent with *
  - Any number of repeated apparences included none

Regex. Example

- Regex to validate e-mail
  - Formed: any letter, digit and _, followed by @
    - ***Regex="***`\w@[a-z]"`
    - string: `marian`@google.com
    - result: n@g
- =>
    - ***Regex="*** \w+@[a-z]+\.[a-z]+ "
    - string: `marian`@google.com
    - result: `marian`@google.com

Python module for regular expressions

- Module **re**
- Usage
  - Define a pattern
    - Pattern = re.compile("regular expression")
  - Use different features
    - findall() – returns a list with all appearances
    - match() – search exact match
    - search() – search a appearance in the string

Python module for regular expressions

- Example
  - Find all dates value from text

text=" Today 3/4/2017 was a day without any tests, next test will be on 4/11/2017"
pattern=re.compile("*\d{1,2}/\d{1,2}/\d{4,4}")*
retVals = pattern.findall(text)

OR

retVals = re.findall(pattren, text)

OR

retVals = re.findall(r"\d{1,2}/\d{1,2}/\d{4,4}", text)