Programming I

Course 5

Introduction to programming

What we discussed about?

Sequences

- Lists
- Tuples

Collections

- Sets
- Dictionaries

What will we talk about?

- Strings
- Formatting Strings
- Regular expressions

Characters

Character representation

- an encoding is used to map a character to its corresponding number

Encodings

- ASCII
  - covers the common Latin characters
- Unicode
  - provide a numeric code for every possible character, in every possible language, on every possible platform

Characters

Functions

- chr() - Converts an integer to a character
- ord() - Converts a character to an integer

Example:

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

Escape characters

Character with special meaning

Formed with \ followed by another character

Examples

- Use " inside a string specified using "
  - "a new \"cuvant\""
- Whitespaces:
  - New line \n : "first line\n second line”
  - Tab \t : “product\tprice”
- Backspace \b

For more details see: https://docs.python.org/2.0/ref/strings.html

Strings

Sequences of characters (like tuples)

Can be defined

- Using " or '

Examples

- 'a valid string', not `an invalid string `
- "a valid string"
- "another' valid string"
- 'another "valid string'
- "not a valid string'

Strings

Sequence-like indexing, slicing

- S="hello”
- S[0] -> h
- S[1] -> e
- S[::-1] -> olleh
- S[1:3] -> el

Operations

- len(S) -> 5

Strings are Immutable

In order to change the content 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:]

String Methods

+ Concatenation

- S1="ab"
- S2="cd"
- S1+S2 -> "abcd”

* Repetitive concatenation

- S1="ab"
- S1*3 -> "ababab"

String Methods

upper()

- Transform a string to upper/lower cases

lower()

- The functions are not destructive (do not change the content of the object)

Examples

S = "Hello"

S.upper() -> "HELLO"

S.lower() -> ”hello"

String Methods

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"]

String Methods

join(sequence)

- returns 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

Formatting Strings

How many string objects are created?

- 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

- helps with lizibility

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 Numbers inside Strings

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

Formatting Strings

Padding

Padding and alignment

```
'{:>10}'.format('test') #align right
'{:10}'.format('test') #align left
'{:_<10}'.format('test') # use _ instead of space
'{:^10}'.format('test') # center
```

Truncating

Truncating long strings

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

Formatting with 'f' or 'r'

f-string	Use f-string instead of format
	name="Ana"
	age="20"
	print(f"Student {name} has {age}.")

r-string	Use r- (raw) to disable escape sequence print(r "ana\nhas\n apples\n") => ana\has\n apples\n

Finding Substrings

- Checking if a substring belongs to a string
  - s.find(x) – returns the index of the fist appearance of x in s
  - x in s

Usual String Operations

Data validation

- A valid e-mail address
- A date is entered in the correct format

Data tokenization

- comma separated values information
- Find the words from a sentence

Regular Expressions - Regex

Compact notation for pattern representation

Defined using a mini-language

Useful for

- Validating data
- Splitting strings after a criteria
- Searching
- Searching and replacing

Regex Characters

Used to search an exact string

Example regex='abc'

- aabcc
- ababc
- ~~aabbcc~~

Special characters

\.^$?+*{}[]()|

Have a significance if they are encountered inside a regex

Example: regex=ab+c

match multile b instances 'b+'

aabcc

ababc

aabbcc

Special characters

\.^$?+*{}[]()|

Have a significance if they are encountered inside a regex

Example: regex=a[bc]d

#match multile b instances 'b' or 'c'

acd

~~abcd~~

~~acbd~~

aabdddd

acdd

Special characters

\.^$?+*{}[]()|

Example

- [0-9] matching a digit
- [^0-9] matching any character that is 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

Using regex for data validation

Calendar dates

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}"

Using regex for data validation

Emails (letters, digist and _, followed by @)

Regex="`\w@[a-z]"`

string: `adi@uvt.ro`

result: `i@u`

Regex - Quantifiers

Syntax: {min,max}

- Sets the number of times the expression is repeating
- Example
  - regex - a{1,1}a{1,1} =>aa == a{1}a{1} == aa
  - regex - a{1,2} =>a or aa

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,} equivalent with +

At least one appearence

{0,} equivalent with *

Any number of repeated appearances including 0

Using regex for data validation

Emails (letters, digist and _, followed by @)

Regex="\w@[a-z]"

string: adi@uvt.ro

result: i@u

Regex=" \w+@[a-z]+\.[a-z]+"

string: adi@uvt.ro

result: adi@uvt.ro

re (regular expressions) module

Define a pattern

pattern = re.compile("regular expression")

Use different features

pattern.findall() – returns a list with all appearances

pattern.match() – search exact match

pattern.search() – search for an appearance in the string

re module

- Find all valid dates from text

```
Text=" Today 26/10/2021 was a day
without any tests, next test will be
on 29/10/2021”

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

OR

retVals = re.findall(pattern, text)
```

Using regex for split

- text = pineapple, apples.grapes; pears   raspberry,strawberry

```
res = text.split(";,. ")
print(res)
=>
['pineapple, apples.grapes; pears        raspberry,strawberry']


import re
res = re.split(r"[;,.\s]\s*", text)
print(res)
=>
['pineapple', 'apples', 'grapes', 'pears', 'raspberry', 'strawberry']
```