Formal Languages and Compilers

Lab1-Introduction

Formal Languages and Compilers Labs

- Content
  - Python – regular expressions
  - FLEX and Bison
- Structure
  - Labs (physical)
  - Small homework
  - 1 big homework – will be verified for cheating

Grades

- 3p lab activity
- 3p homework (it will be checked during the lab; if you cannot attend the lab, send it to me via email prior to the beginning of it)
- 4p big homework
- Late submissions are not accepted

- Condition for passing
  - Attendance: min. 7 labs
  - Grade: minimum 1.5p/3 lab, 1.5p/3 hw & 2p/4 big hw

Questions

If you have any questions, please feel free to contact me at

ciungan_alexandra@yahoo.ro

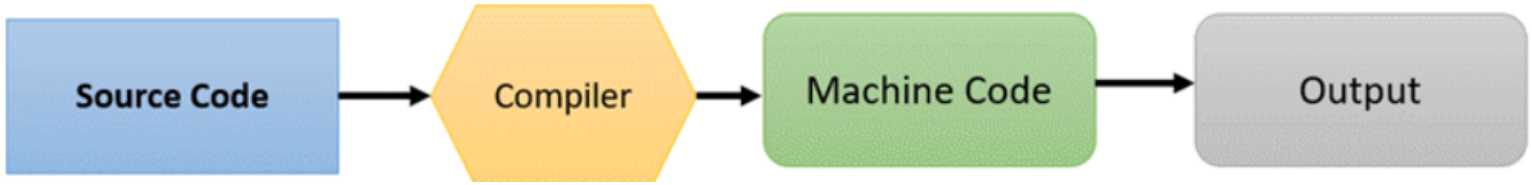
Feedback

- If you have any kind of feedback during the semester, please send it at:

https://forms.gle/RNGVe3bjqB2wFpo59

Formal Languages and Compilers -Theoretical Notions-

- Compiler: computer software that converts computer code written in one programming language into another one (all at once).



- Interpreter: computer program that converts each statement during run

Compiled language vs Interpreted language

Criteria	Compiled Language	Interpreted Language
Programming language	The implementations are typically compilers and not interpreters.	The implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions.
Running a program	Once the program is compiled it is expressed in the instructions of the target machine.	The instructions are not directly executed by the target machine.
Steps	There are at least two steps to get from source code to execution.	There is only one step to get from source code to execution.
Speed	Compiled programs run faster	Although slower, interpreted programs can be modified while the program is running.
Errors	Compilation errors prevent the code from compiling.	All the debugging occurs at run-time
Examples	C, C++, C#	Python, JavaScript, Perl

Python

- Interpreted language (execute instructions directly and freely, without previously compiling a program into machine-language instructions)
- One of the most popular programming language

- Used for: web development (server-side), software development, system scripting, mathematics, task automatization, data analysis
- Simple syntax
- Can be used on Windows, Mac, Linux, Raspberry Pi etc.

- Syntax: Indentation is highly important (it is used to indicate a block of code)
- print(): function that prints “messages”, variables; standard output device

```
a="world"
print("Hello, "+ a)
```

```
In [1]: runfile('D:/Facultate/Predat/FormalLanguagesAndCompilers/Lab/Ex/example.py', wdir='D:/
Facultate/Predat/FormalLanguagesAndCompilers/Lab/Ex')
Hello world

In [2]:
```

Python
-Data Types-

Data Type	Example	Type
String	x = “Hello!”	Text
Integer	x = 7	Numeric
Float	x = 7.3	Numeric
Complex	x =1j	Numeric
List	x = [“student”, “professor”]	Sequence
Tuple	x = (“student”, “professor”)	Sequence
Range	x = range(5)	Sequence
Dictionary	x = {“name”:”Jane”, “age”:21}	Map
Set	x = {“student”, “professor”}	Set
Frozenset	x = frozenset({“student”, “professor”})	Set
Boolean	x = True	Boolean
Bytes	x = b”Hello”	Binary
Bytearray	x = bytearray(5)	Binary
Memoryview	x = memoryview(bytes(6))	Binary

Python
-Variables-

- Create a variable by assigning a value to it (e.g., x=7), you do not need to declare a type to it
- If the same variable is declared twice, then the latter value will be the one assigned (e.g., x=7 x=7.5; now x is a float and has the value 7.5)
- Casting: you can cast a value if you want to specify the data type (e.g., x=int(7))
- Strings: you can use ‘ or “” to declare strings (thus, x=“Jane” is the same with x=‘Jane’)
- Variable names are case-sensitive (thus, a=3 and A=5 are different), start with a letter or _ (NOT with a number) and can contain only letters (A-Z, a-z), numbers and underscores

Python
-Variable's declaration using casting-

DataType	Example
String	x = str(“Hello!”)
Integer	x = int(7)
Float	x = float(7.3)
Complex	x =complex(1j)
List	x = list([“student”, “professor”])
Tuple	x = tuple((“student”, “professor”))
Range	x = range(5)
Dictionary	x = dict({“name”:”Jane”, “age”:21})
Set	x = set({“student”, “professor”})
Frozenset	x = frozenset({“student”, “professor”})
Boolean	x = bool(True)
Bytes	x = b”Hello”
Bytearray	x = bytearray(5)
Memoryview	x = memoryview(bytes(6))

Python
-Variables-

- You can declare variables one by one
```
x=7
y=7.8
```
or you can assign multiple va
```
x , y =7 , 7.8
```
ne, or you can assign the same valu
```
x = y =7
```
same variables in one line
- You can add variables of the same data type using +

Code

Ou

```
x = "I am a "
y = "student at FILS"
z=x+y

print(x+y, " --- " , z)
```

```
I am a student at FILS  ---  I am a student at FILS
```

```
x = "I am a "
y = "student at FILS"
z=x+y

print(x+y, "\n" , z)
```

```
I am a student at FILS
 I am a student at FILS
```

Python
-range-

- Creates a sequences of numbers, starting from 0 and incremented by 1 (by default), until the number specified – range(stop)
  - range(3) - > 0,1,2
- range(start, stop, step) - you can specify the start and end of the sequences and, also, the incrementation wished (the end of the sequence stops at -1 from the value specified)
  - range(3,8) -> 3,4,5,6,7
  - range(3,11,2) -> 3,5,7,9

Python
-List-

- Store multiple items in one variable
- Changeable
- Indexed (first index: 0)
- Allows duplicates
- len() – length of the list
- You can unpack a list (extract values into variables)
- Python does not have arrays, but lists can be used as a replacemen

```
mylist=[ "Rocket League", "FIFA 2021", "Horizon 0 Dawn"]
a,b,c = mylist

print(a)
print("\n",b)
print("\n",c)
```

```
Rocket League

 FIFA 2021

 Horizon 0 Dawn
```

Python
-List-

- Simple example

```
mylist=[ "Rocket League", "FIFA 2021", "Horizon 0 Dawn", "Control", "Borderlands",
        "Minecraft", "Terraria", "Need for Speed Heat", "Mass Effect", "Fallout New Vegas"]

print("This is the last element of the list:", mylist[-1])
print("This is the second element of the list:", mylist[1])
print("This is the element with index 4:", mylist[4])
#the last element is not included
print("These are the second, third and fourth elements of the list", mylist[1:4])
print("These are the first 3 elements of the list", mylist[:3])
print("These are the last 3 elements of the list", mylist[7:])
print("The length of the list is:", len(mylist))
```

```
This is the last element of the list: Fallout New Vegas
This is the second element of the list: FIFA 2021
This is the element with index 4: Borderlands
These are the second, third and fourth elements of the list ['FIFA 2021', 'Horizon 0 Dawn', 'Control']
These are the first 3 elements of the list ['Rocket League', 'FIFA 2021', 'Horizon 0 Dawn']
These are the last 3 elements of the list ['Need for Speed Heat', 'Mass Effect', 'Fallout New Vegas']
The length of the list is: 10
```

Python
-List Methods-

Method	Description	Example x=[1,3,2,3, 4]	Output
append()	Adds an element at the end of the list	x.append(5)	x = [1, 3, 2, 3, 4, 5]
clear()	Removes all the elements from the list	x.clear()	x = []
copy()	Returns a copy of the list	z=x.copy()	z = [1, 3, 2, 3, 4]
count()	Returns the number of elements with the specified value	z=x.count(3)	z = 2
extend()	Add the elements of a list (or any iterable) to the end of the current list	z=[7,6] x.extend(z)	x = [1, 3, 2, 3, 4, 7, 6]
index()	Returns the index of the first element with the specified value	z=x.index(2)	z = 2
insert()	Adds an element at the specified position	x.insert(2,7)	x = [1, 3, 7, 2, 3, 4]
pop()	Removes the element at the specified position	x.pop(3)	x = [1, 3, 2, 4]
remove()	Removed the item with the specified value	x.remove(1)	x = [3, 2, 3, 4]
reverse()	Reverses the order of the list	x.reverse()	x = [4, 3, 2, 3, 1]
sort()	Sorts the list	x.sort()	x = [1, 2, 3, 3, 4]

Python
-Tuple-

- Store multiple items in one variable
- Unchangeable
- Ordered
- Can contain items of different types
- Simple example

```
myTuple=("Hello", 3, False, [1,4,6])

print(myTuple)
```

```
('Hello', 3, False, [1, 4, 6])
```

Python
-Tuple-

- Simple example with methods

```
myTuple=(2,3,4,1,8,6,7,6)
mylist=[1,4,3]

#convert list into tuple
myTuple2=tuple(mylist)
myTuple3=myTuple+myTuple2

print("The item with index 2 is:", myTuple[2])
print("The length of the tuple is:",len(myTuple))
print("The item with the highest value is:", max(myTuple))
print("The item with the lowest value is:", min(myTuple))
print("The sum of all the items is:", sum(myTuple))
print("The type of myTuple2 is:", type(myTuple2))
print("The tuple is sorted from lowest to highest:", sorted(myTuple))
print("The value 6 appears in the tuple this amount of times:", myTuple.count(6))
print("The concatenation of myTuple and myTuple2 is:", myTuple3)
```

```
The item with index 2 is: 4
The length of the tuple is: 8
The item with the highest value is: 8
The item with the lowest value is: 1
The sum of all the items is: 37
The type of myTuple2 is: <class 'tuple'>
The tuple is sorted from lowest to highest: [1, 2, 3, 4, 6, 6, 7, 8]
The value 6 appears in the tuple this amount of times: 2
The concatenation of myTuple and myTuple2 is: (2, 3, 4, 1, 8, 6, 7, 6, 1, 4, 3)
```

Python
-Dictionary-

- Store values in key:value pairs
- Ordered (starting Python 3.7), changeable, does not allow duplicates
- Unique keys, values can be of any data type
- Simple example

```
myFirstDictionary = {
  "bookName": "Perfume: The Story of a Murderer",
  "bAuthor": "Patrick Suskind",
  "year": 1985,
  "inStock": False,
  "otherBooksFromSameAuthor": ["The Double Bass", "The Pigeon", "On Love and Death"]
}

print(myFirstDictionary)
```

```
{'bookName': 'Perfume: The Story of a Murderer', 'bAuthor': 'Patrick Suskind', 'year': 1985,
'inStock': False, 'otherBooksFromSameAuthor': ['The Double Bass', 'The Pigeon', 'On Love and
Death']}
```

Python
-Dictionary Methods-

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If it does not exist, it will insert the key with the specified value
update()	Update the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Exercises

I. list1 = [“Math”, “English”, “History”, “Chemistry”, “Biology”]
   1. Print the second item in the list
   2. Print the number of items in the list
   3. Print the second, third and forth items in the list, using their index
   4. Print the fourth element from the end of the list
   5. Copy the list into a new variable and remove “Chemistry” from the new list
   6. Copy the list into a new variable and add “Geography” as the third element of the new list
   7. Copy the list into a new variable and change “English” to “Romanian” in the new list

II. dict1 = {“Fname” : ”Jane”, “Lname” : ”Doe”, “age” : 23}
   1. Print the value for “Lname”
   2. Copy the dictionary into a new variable and change the age from 23 to 21 in the new dictionary
   3. Add the key/value pair “occupation” : “student” to the dictionary
   4. Copy the dictionary into a new variable and remove Fname from the new dictionary
   5. Print a list that contains all the key/value pairs of the dictionary
   6. Print a list that contains all the values in the dictionary
   7. Use setdefault() to search for “hobby” : “chess” and print the updated dictionary
   8. Empty the dictionary

Python
-Installation-

- For next week, please install Python 3 (at least version 3.8)

- Can be installed in VSC (Visual Studio Code) via the Python extension
- Can be used in Anaconda
- Can be used in PyCharm