Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

facebook.com/cs.MohamedAyman

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 1

Python Overview

Course Roadmap

Part 1: Python Basics and Functions

Lecture 1: Python Overview

Lecture 2: Variable Types

Lecture 3: Basic Operations

Lecture 4: Conditions

Lecture 5: Loops

Lecture 6: Functions

Lecture Agenda

We will discuss in this lecture the following topics

1- History of Python

2- Interpreter vs. Compiler

3- Python Identifiers & Reserved Words

4- Lines and Indentation

5- Multi-Line Statements

6- Quotation & Comments


Let's
START UP

Lecture Agenda

Section 1: History of Python

Section 2: Interpreter vs. Compiler

Section 3: Python Identifiers & Reserved Words

Section 4: Lines and Indentation

Section 5: Multi-Line Statements

Section 6: Quotation & Comments

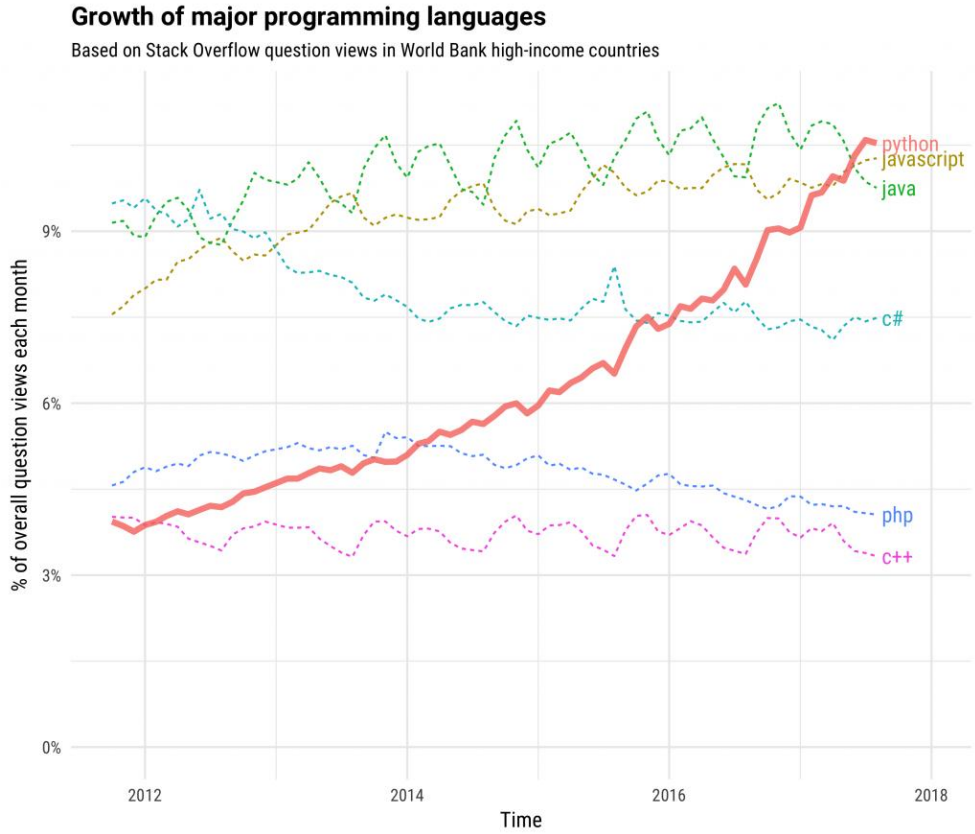
History of Python

- Python was developed by Guido van Rossum in the late eighties and early nineties at the national Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License.
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.
- Python 1.0 was released in November 1994. In 2000, Python 2.0 was released. Python 2.7.11 is the latest edition of Python 2.
- Meanwhile, Python 3.0 was released in 2008. Python 3 is not backward compatible with Python 2. The emphasis in Python 3 has been on the removal of duplicate programming constructs and modules so that "There should be one -- and preferably only one -- obvious way to do it." Python 3.8 is the latest version of Python 3.

History of Python

- Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas the other languages use punctuations. It has fewer syntactical constructions than other languages.
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language: Python is a great language for the beginner level programmers and suppots the development of a wide range of applications from simple text processing to WWW browers to games.

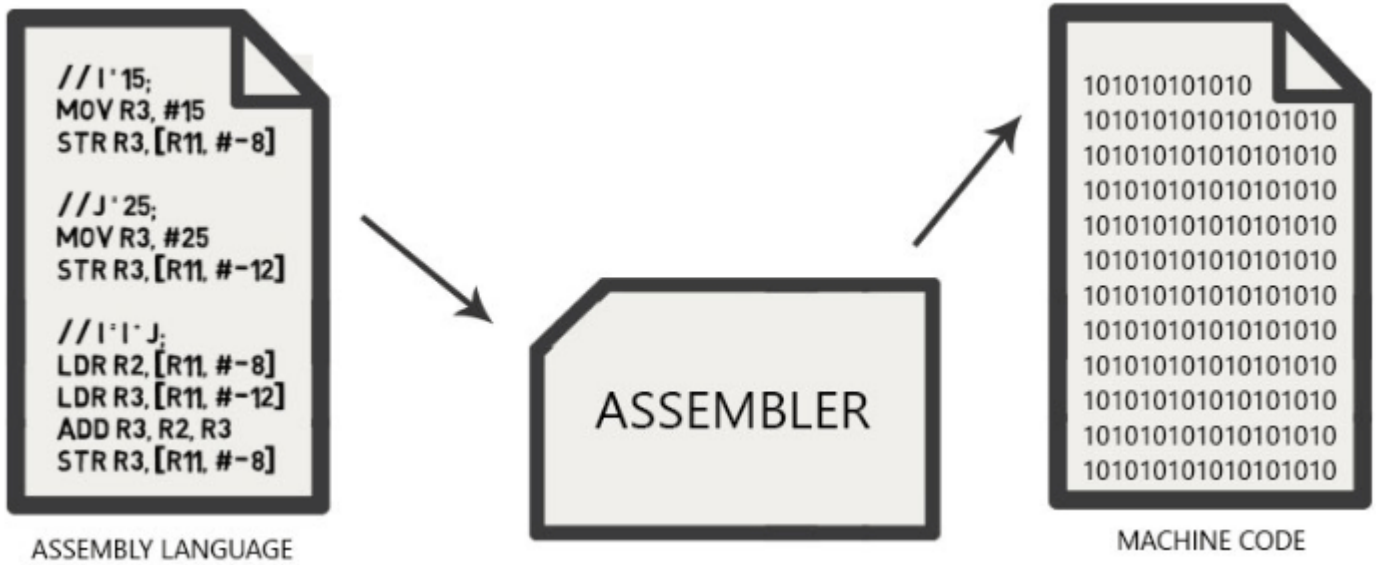
History of Python



Lecture Agenda

- ✔ Section 1: History of Python
- **Section 2: Interpreter vs. Compiler**
- Section 3: Python Identifiers & Reserved Words
- Section 4: Lines and Indentation
- Section 5: Multi-Line Statements
- Section 6: Quotation & Comments

Low-level languages

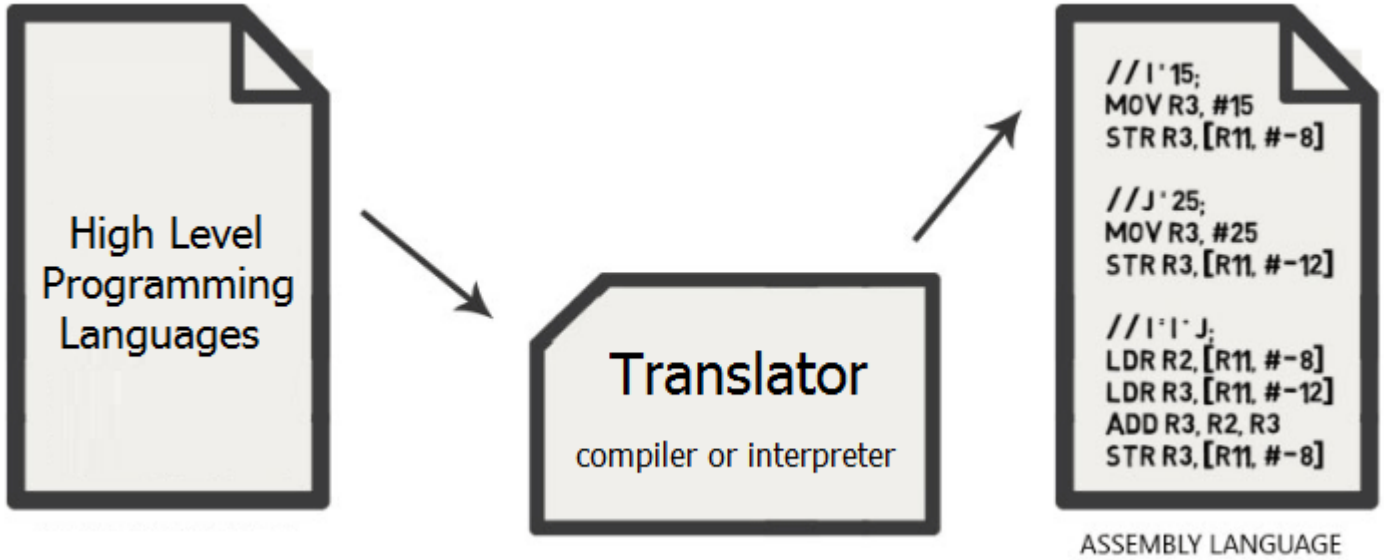


- machine oriented and require extensive knowledge of computer hardware and its configuration

1- Machine language: language that is directly understood by the computer, and it does not need to be translated

2- Assembly language: set of symbols and letters

High-level languages



- programming language that uses English and mathematical symbols in its instructions

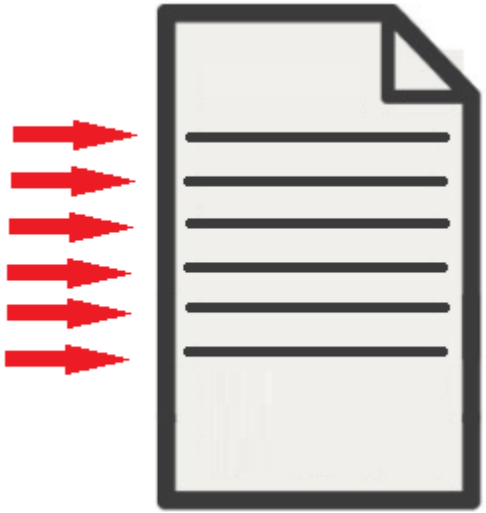
1- Compiled: computer program that translates a program written in a high-level language to the machine language of a computer

2- Interpreted: computer program that simulates a computer that understands a high-level language

Compiler

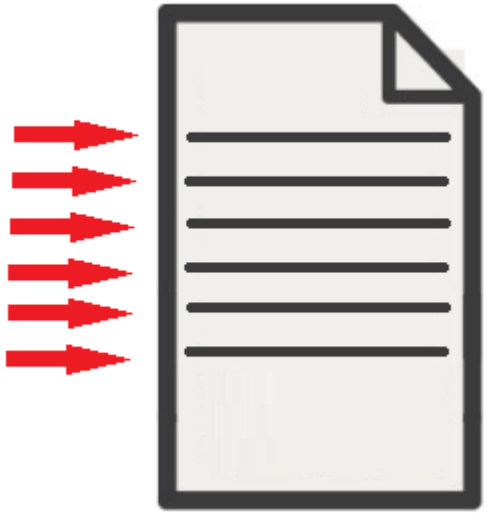
Phase 1:

(check syntax & semantic for each statement)

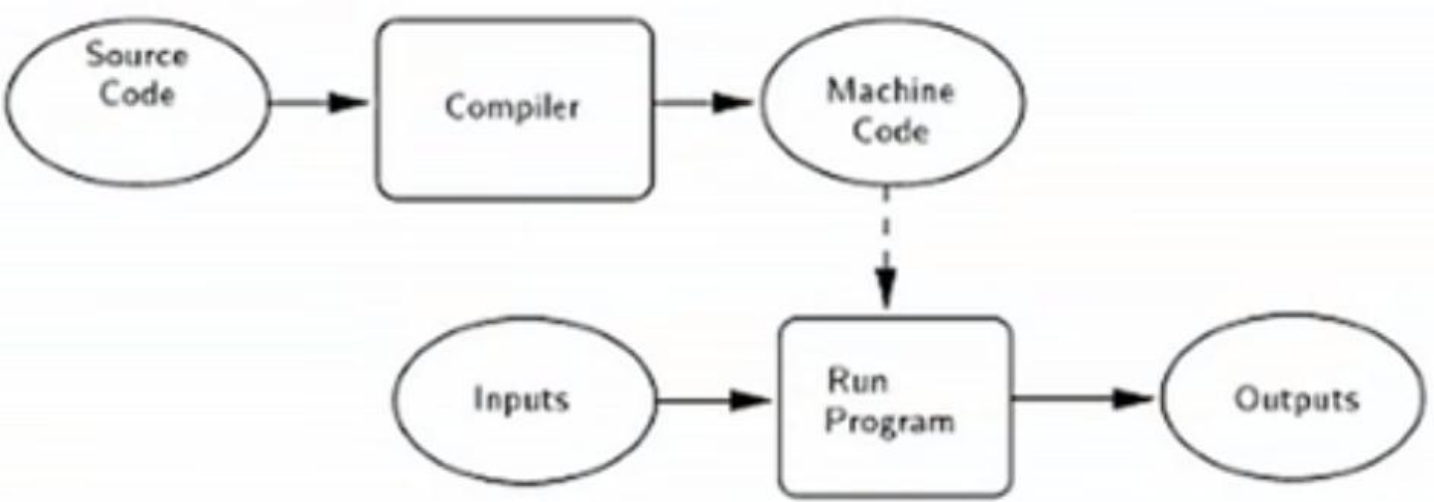


Phase 2:

(execute each statement)



Compiler



Pros:

1- ready to run

2- often faster

3- source code is private

Cons:

1- not cross-platform

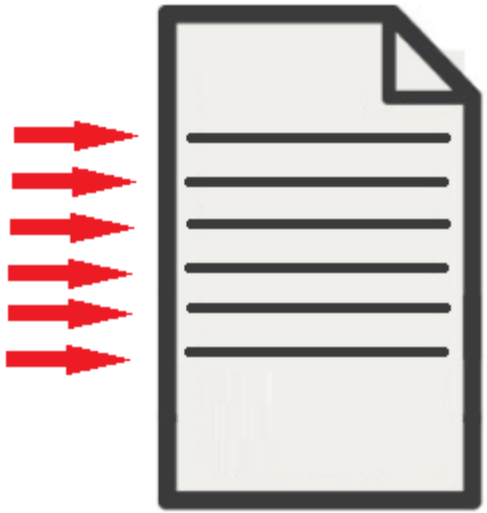
2- inflexible

3- extra step

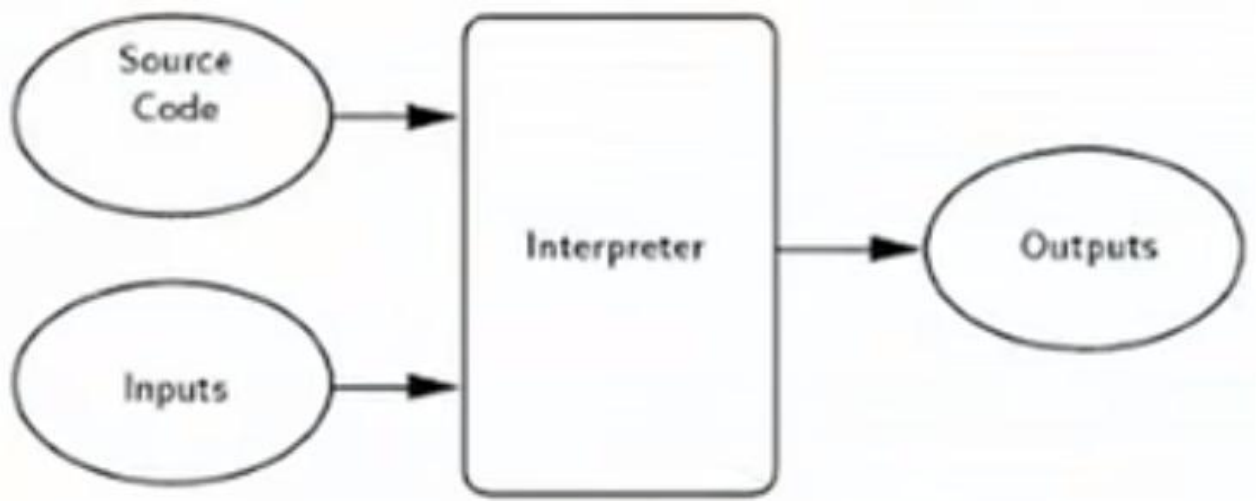
Interpreter

Phase 1 & 2:

(check syntax & semantic for each statement and execute it)



Interpreter



Pros:

1- cross plat-form

2- simpler to test

3- easier to debug

Cons:

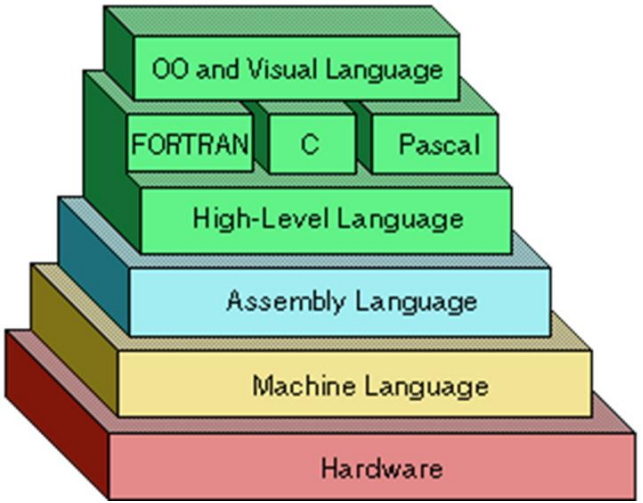
1- interpreter required

2- often slower

3- source code is public

Levels of programming languages

- Python is a high-level language intended to be relatively straightforward for humans to read and write and for computers to read and process. Other high-level languages include Java, C++, PHP, Ruby, Basic, Perl, JavaScript, and more.
- The actual hardware inside the Central Processing Unit (CPU) does not understand any of these high-level languages. The CPU understands a language we call machine language. Machine language is very simple and frankly very tiresome to write because it is represented all in zeros and ones.
- Machine language seems quite simple on the surface, given that there are only zeros and ones, but its syntax is even more complex and far more intricate than Python. So very few programmers ever write machine language. Instead we build various translators to allow programmers to write in high-level languages like Python or JavaScript and these translators convert the programs to machine language for actual execution by the CPU.



Lecture Agenda

- ✔ Section 1: History of Python
- ✔ Section 2: Interpreter vs. Compiler
- **Section 3: Python Identifiers & Reserved Words**
- Section 4: Lines and Indentation
- Section 5: Multi-Line Statements
- Section 6: Quotation & Comments

Python Program

- The definition of a program at its most basic is a sequence of Python statements that have been crafted to do something. Even our simple hello.py script is a program. It is a one-line program and is not particularly useful, but in the strictest definition, it is a Python program.
- It might be easiest to understand what a program is by thinking about a problem that a program might be built to solve, and then looking at a program that would solve that problem.
- Typing commands into the Python interpreter is a great way to experiment with Python's features, but it is not recommended for solving more complex problems.
- When we want to write a program, we use a text editor to write the Python instructions into a file, which is called a script. By convention, Python scripts have names that end with .py.

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other objects.
- An identifier starts with a letter [A to Z] or [a to z] or an underscore [_] followed by zero or more letters [A to Z] or [a to z], underscore [_] and digits [0 - 9].
- Python does not allow punctuation characters such as !, @, #, $, and % within identifiers.
- Python is a case sensitive programming language. Thus, Var and var are two different identifiers.
- Here are naming conventions for Python identifiers
  - Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
  - Starting an identifier with a single leading underscore indicates that the identifier is private.
  - Starting an identifier with two leading underscores indicates a strong private identifier.
  - If the identifier also ends with two trailing underscores, the identifier is a language defined special name.

Quiz

- Which of the following variables is valid, according to python identifier?

- [ ] `xyz = 'string value'`
- [ ] `abc = 1634`
- [ ] `@url_link = 'www.google.com'`
- [ ] `var1 = 637 + 814`
- [ ] `_ = 'temp value'`
- [ ] `_var2 = "be careful for this variable name"`
- [ ] `MIN_NUM = -1000000000`
- [ ] `max_length = 100000`
- [ ] `2var = 2 * var1`
- [ ] `address%value = '123 block #7, floor #4, Berland'`

Quiz Solution

- Which of the following variables is valid, according to python identifier?

- [x] `xyz = 'string value'`
- [x] `abc = 1634`
- [ ] `@url_link = 'www.google.com'`
- [x] `var1 = 637 + 814`
- [x] `_ = 'temp value'`
- [x] `_var2 = "be careful for this variable name"`
- [x] `MIN_NUM = -1000000000`
- [x] `max_length = 100000`
- [ ] `2var = 2 * var1`
- [ ] `address%value = '123 block #7, floor #4, Berland'`

Reserved Words

- The following list shows the Python keywords. These are reserved words and you cannot use them as constants or variables or any other identifier name.
- All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

Keywords in Python programming language

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Quiz

- Which of the following variables is valid, according to python identifier and python reserved words?

- [ ] `xyz = 'string value'`
- [ ] `abc = 1634`
- [ ] `if = 'www.google.com'`
- [ ] `var1 = 637 + 814`
- [ ] `_ = 'temp value'`
- [ ] `_var2 = "be careful for this variable name"`
- [ ] `MIN_NUM = -1000000000`
- [ ] `max_length = 100000`
- [ ] `for = 2 * var1`
- [ ] `break = '123 block #7, floor #4, Berland'`

Quiz Solution

- Which of the following variables is valid, according to python identifier and python reserved words?

- [x] `xyz = 'string value'`
- [x] `abc = 1634`
- [ ] `if = 'www.google.com'`
- [x] `var1 = 637 + 814`
- [x] `_ = 'temp value'`
- [x] `_var2 = "be careful for this variable name"`
- [x] `MIN_NUM = -1000000000`
- [x] `max_length = 100000`
- [ ] `for = 2 * var1`
- [ ] `break = '123 block #7, floor #4, Berland'`

Lecture Agenda

- ✔ Section 1: History of Python
- ✔ Section 2: Interpreter vs. Compiler
- ✔ Section 3: Python Identifiers & Reserved Words
- **Section 4: Lines and Indentation**
- Section 5: Multi-Line Statements
- Section 6: Quotation & Comments

Lines and Indentation

- Python does not use braces { } to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

- However, the following block generates an error Thus, in Python all the continuous lines indented with the same number of spaces would form a block. The following example has various statement blocks.

Quiz

- Which of the following blocks is valid, according to python indentation?

```python
N = int(5e3+3)
n = int(input())
res = [0] * N
cum = list([0] * N for i in range(N))
a = list(map(int, input().split()))
for i in range(n):
    c = [0] * N
    for j in range(i, n):
        c[a[j]] += 1
        cum[i][j] = c[a[j]]
for i in range(n):
    curr_max, curr_idx = 0, N-1
    j = 0
    while cum[i][j] == 0:
        j += 1
    while j < n:
        if cum[i][j] > curr_max:
            curr_max = cum[i][j]
            curr_idx = a[j]
        res[curr_idx] += 1
        j += 1
print(*res[1:n+1])
```

```python
N = int(5e3+3)
n = int(input())
res = [0] * N
cum = list([0] * N for i in range(N))
a = list(map(int, input().split()))
for i in range(n):
    c = [0] * N
    for j in range(i, n):
            c[a[j]] += 1
            cum[i][j] = c[a[j]]
for i in range(n):
    curr_max, curr_idx = 0, N-1
    j = 0
    while cum[i][j] == 0:
            j += 1
    while j < n:
        if cum[i][j] > curr_max:
                curr_max = cum[i][j]
                curr_idx = a[j]
        res[curr_idx] += 1
        j += 1
print(*res[1:n+1])
```

Quiz Solution

- Which of the following blocks is valid, according to python indentation?

```python
N = int(5e3+3)
n = int(input())
res = [0] * N
cum = list([0] * N for i in range(N))
a = list(map(int, input().split()))
for i in range(n):
    c = [0] * N
    for j in range(i, n):
        c[a[j]] += 1
        cum[i][j] = c[a[j]]
for i in range(n):
    curr_max, curr_idx = 0, N-1
    j = 0
    while cum[i][j] == 0:
        j += 1
    while j < n:
        if cum[i][j] > curr_max:
            curr_max = cum[i][j]
            curr_idx = a[j]
        res[curr_idx] += 1
        j += 1
print(*res[1:n+1])
```

```python
N = int(5e3+3)
n = int(input())
res = [0] * N
cum = list([0] * N for i in range(N))
a = list(map(int, input().split()))
for i in range(n):
    c = [0] * N
    for j in range(i, n):
            c[a[j]] += 1
            cum[i][j] = c[a[j]]
for i in range(n):
    curr_max, curr_idx = 0, N-1
    j = 0
    while cum[i][j] == 0:
            j += 1
    while j < n:
        if cum[i][j] > curr_max:
                curr_max = cum[i][j]
                curr_idx = a[j]
        res[curr_idx] += 1
        j += 1
print(*res[1:n+1])
```

Types of Error

- Syntax errors:

These are the first errors you will make and the easiest to fix. A syntax error means that you have violated the “grammar” rules of Python. Python does its best to point right at the line and character where it noticed it was confused. The only tricky bit of syntax errors is that sometimes the mistake that needs fixing is actually earlier in the program than where Python noticed it was confused. So the line and character that Python indicates in a syntax error may just be a starting point for your investigation.

```
>>> print('Hello World)
SyntaxError: EOL while scanning string literal
```

Types of Error

- Runtime errors:

The second type of error is a runtime error, so called because the error does not appear until after the program has started running. These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened. Runtime errors are rare in the simple programs you will see in the first few chapters, so it might be a while before you encounter one.

```
>>> 5 / 0
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    5 / 0
ZeroDivisionError: division by zero
```

Types of Error

- Semantic errors:

The third type of error is “semantic”, which means related to meaning. If there is a semantic error in your program, it will run without generating error messages, but it will not do the right thing. It will do something else. Specifically, it will do what you told it to do. Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

```
>>> 'Hello' + 3
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    'Hello' + 3
TypeError: Can't convert 'int' object to str implicitly
```

Lecture Agenda

- ✔ Section 1: History of Python
- ✔ Section 2: Interpreter vs. Compiler
- ✔ Section 3: Python Identifiers & Reserved Words
- ✔ Section 4: Lines and Indentation
- **Section 5: Multi-Line Statements**
- Section 6: Quotation & Comments

Multi-Line Statements

- Statement in Python typically end with a new line. Python, however, allows the use of the line continues character ( ) to denote that the line should continue. The Statement contained within the [ ], { }, or ( ) brackets do not need to use the line continuation character Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
>>> days = ['Sunday', 'Monday', 'Tuesday',
        'Wednesday', 'Thursday', 'Friday',
        'Saturday']
>>> print(days)
['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']
```

Multi-Line Statements

- Statement in Python typically end with a new line. Python, however, allows the use of the line continues character ( ) to denote that the line should continue. The Statement contained within the [ ], { }, or ( ) brackets do not need to use the line continuation character Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
>>> first_num = 5
>>> second_num = 7
>>> third_num = 1
>>> forth_num = 2
>>> total = first_num + second_num + \
        third_num + forth_num + \
        10 + 100
>>> print(total)
125
```

Multi-Line Statements

- The semicolon (;) allows multiple statement on a single line given that no statement starts a new code block. We could also put multiple statements in a single line using semicolons, as follows

```
>>> a = 1 ; b = 2 ; c = a + b
>>> print(a) ; print(b) ; print(c)
1
2
3
```

Lecture Agenda

- ✔ Section 1: History of Python
- ✔ Section 2: Interpreter vs. Compiler
- ✔ Section 3: Python Identifiers & Reserved Words
- ✔ Section 4: Lines and Indentation
- ✔ Section 5: Multi-Line Statements
- **Section 6: Quotation & Comments**

Python Quotation

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes are used to span the string across multiple lines. For example, all the following are legal

```
>>> x = 'word'
>>> print(x)
word
>>> y = "This is a sentence"
>>> print(y)
This is a sentence
>>> z = '''This is a paragraph, It is made
up of multiple lines and sentence.'''
>>> print(z)
This is a paragraph, It is made
up of multiple lines and sentence.
```

Python Comments

- Python developers often make use of the comment system as, without the use of it, things can get real confusing, real fast. Comments are the useful information that the developers provide to make the reader understand the source code. It explains the logic or a part of it used in the code. Comments are usually helpful to someone maintaining or enhancing your code when you are no longer around to answer questions about it. These are often cited as a useful programming convention that does not take part in the output of the program but improves the readability of the whole program. There are two types of comment in Python:

1- Single line comments

2- Multi-line string as comment

Python Comments

- Single line comments:

Python single line comment starts with hashtag symbol with no white spaces (#) and lasts till the end of the line. If the comment exceeds one line then put a hashtag on the next line and continue the comment. Python's single line comments are proved useful for supplying short explanations for variables, function declarations, and expressions. See the following code snippet demonstrating single line comment:

Example:

```
# This is a comment
# Print "Hello World!" to console
print("Hello World!")
```

Example:

```
a, b = 1, 3 # Declaring two integers
sum = a + b # adding two integers
print(sum) # displaying the output
```

Python Comments

- Multi-line string as comment:

Python multi-line comment is a piece of text enclosed in a delimiter (""") on each end of the comment. Again there should be no white space between delimiter """). They are useful when the comment text does not fit into one line; therefore needs to span across lines. Multi-line comments or paragraphs serve as documentation for others reading your code. See the following code snippet demonstrating multi-line comment:

Example:

```
"""
This would be a multiline comment in Python that spans several lines and describes.
A Computer Science portal. It contains well written, well thought and
well-explained computer science and programming articles, quizzes and more.
"""
print("Hello World!")
```

Example:

```
'''
This example gives you a
perfect example of multi-line comments
'''
print("Hello World!")
```

Lecture Agenda

- ✓ Section 1: History of Python
- ✓ Section 2: Interpreter vs. Compiler
- ✓ Section 3: Python Identifiers & Reserved Words
- ✓ Section 4: Lines and Indentation
- ✓ Section 5: Multi-Line Statements
- ✓ Section 6: Quotation & Comments


DO
MORE.