Introduction to Programming and Problem Solving

Week 3: Basics of Python Programming

Lecturer: Lemi Agrey Oliver

Department of Information Technology

Kumi University

Email: codingissweet@gmail.com

Recap for last week

- Last week, we delved into the interesting world of problem-solving, exploring its various features and complexities. We commenced by delving into the fundamental concept of problem analysis, examining the diverse features and characteristics that define a problem.
- Our exploration continued as we dissected the crucial steps involved in problem analysis, offering a comprehensive understanding of the processes and methodologies employed to unravel complex issues.

Recap for last week+

- We also dedicated time to explore problem-solving methodologies, shedding light on the systematic approaches used to tackle challenges effectively.

1. Furthermore, we delved into the area of visual problem representation, discussing essential tools such as flowcharts, structure charts, and pseudocode.

- These visual aids play a pivotal role in breaking down complex problems into manageable components, aiding in the development of efficient solutions.

Recap for last week++

- In addition to these visual aids, we embarked on a deep dive into the concept of algorithms. We examined the various types of algorithms and their significance in the world of computer science. Algorithms serve as the backbone of problem-solving in the digital realm, providing structured sequences of instructions to efficiently address a wide range of challenges
- This week we shall embark on yet an exciting journey into Python programming.

Recap for last week +++

- Python is a powerful tool widely utilized by computer scientists to harness the capabilities of computers in solving intricate problems. It offers a versatile and user-friendly platform for coding and automation, making it a valuable skill in the world of technology and problem-solving

Content

- Brief history of python
- Why python
- Python Installation on windows
- Datatypes
- Variable
- Formatting Number and Strings
- Python inbuilt Functions

Introduction to Python

- Python is a simple, general purpose, interpreted, high level, and object-oriented programming language.
- *It was founded by Guido Van Rossum* in 1989 in the Netherland
- The name python was driven from a popular comedy show series of 1970s that was aired on BBC .
- In February 1991, **Guido Van Rossum** published the code (labeled version 0.9.0) to alt . sources.

Introduction to Python+

- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
- Python 2.0 added new features such as list comprehensions, garbage collection systems.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language

Why Python

- **Ease of Learning,** Python's syntax is straightforward and easy to read, making it an ideal language for beginners. Its simplicity allows newcomers to focus on learning programming concepts without getting bogged down in complex syntax.
- **Versatility,** Python is a general-purpose programming language, meaning it can be used for a wide range of applications. It can be used in web development, data analysis, machine learning, scientific computing, or automation.

Why Python+

- **Large and Active Community,** Python has a vast and active community of developers who contribute to its growth. This community provides extensive documentation, tutorials, and support, making it easier for programmers to find solutions to their problems.
- **Rich Standard Library,** Python comes with a comprehensive standard library that includes modules and packages for various tasks. This extensive library simplifies common programming tasks, reducing the need to write code from scratch.

Why Python..

- **Third-Party Libraries and Frameworks,** Python boasts a rich ecosystem of third-party libraries and frameworks. Libraries like NumPy, pandas, TensorFlow, Django, Flask, and many others extend Python's capabilities for specialized domains. This abundance of resources saves developers time and effort.

- **Cross-Platform Compatibility,** Python is available for various operating systems, including Windows, macOS, and Linux. This cross-platform compatibility ensures that Python code can be easily executed on different systems without modification.

Why Python…

- **Interpreted Language,** Python is an interpreted language, which means you can run your code without the need for a separate compilation step. This allows for rapid development and easy debugging.
- **Dynamic Typing,** Python is dynamically typed, so you don't need to declare variable types explicitly. This flexibility simplifies coding and allows for more rapid development.

Why Python….

- **Open Source,** Python is open-source and free to use, making it accessible to everyone. It's continually evolving, with contributions from developers worldwide.
- **Data Science and Machine Learning,** Python has become the language of choice for data scientists and machine learning engineers. Its libraries, such as NumPy, pandas, scikit-learn, and TensorFlow, provide powerful tools for data analysis, modeling, and machine learning.

Why Python…..

- **Scalability,** Python is used by both startups and large-scale enterprises.

Its scalability and robustness are demonstrated by its use in a variety of applications, from small scripts to complex systems.

Why Python……

- **Readability and Maintainability,** Python's emphasis on code readability and clean, consistent formatting (using indentation) promotes good coding practices. This makes it easier for teams to collaborate and maintain codebases.
- **Scripting and Automation,** Python is a powerful scripting language. It excels at automating repetitive tasks, making it a valuable tool for system administrators and DevOps professionals.

Python Usage

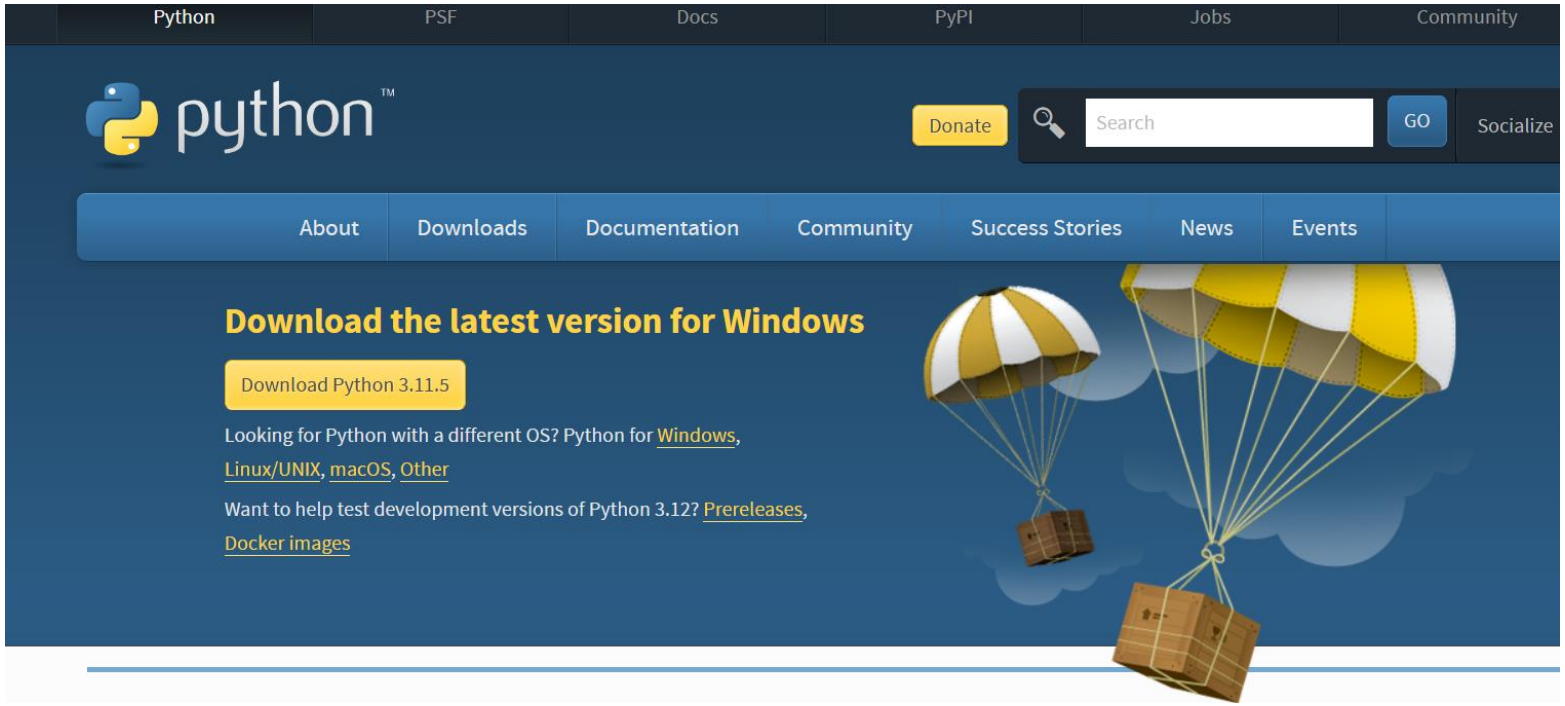
- **Web Development,** Python is used to build web applications using frameworks like Django, Flask, and Pyramid.
- **Data Science and Analysis,** Python is widely used in data science and analytics with libraries like NumPy, pandas, and Matplotlib.
- **Machine Learning and AI,** Libraries such as TensorFlow, PyTorch, and scikit-learn make Python a popular choice for machine learning and artificial intelligence.

Python Usage.

- **Scripting,** Python is often used for scripting tasks, automation, and system administration.
- **Game Development,** Python is used in game development with libraries like Pygame.
- **Scientific Computing,** Python is favored in scientific and engineering applications.
- **Desktop Applications,** Tools like PyQt and Tkinter allow developers to create desktop applications with Python.

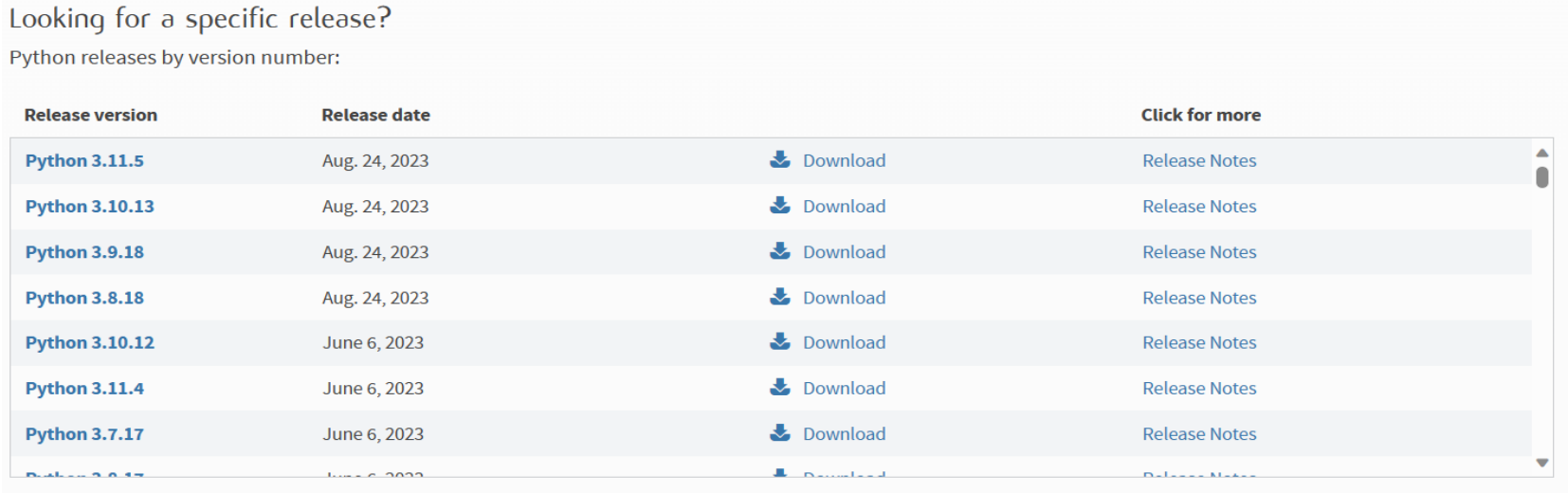
Installing Python on widows

- Python installation on windows is straight forward and easy
- First you have to download the python software from [Download Python | Python.org]
- Then you will see the below screen



Installing Python on widows.

- Then click download python 3.11.5 which is the latest version of python by the time of the preparation of this lecture however by the time of your interaction with this materials there might be a newer version of python to download.
- Or you might also want to install an older version of python, in that case you scroll down the page and select the version you want



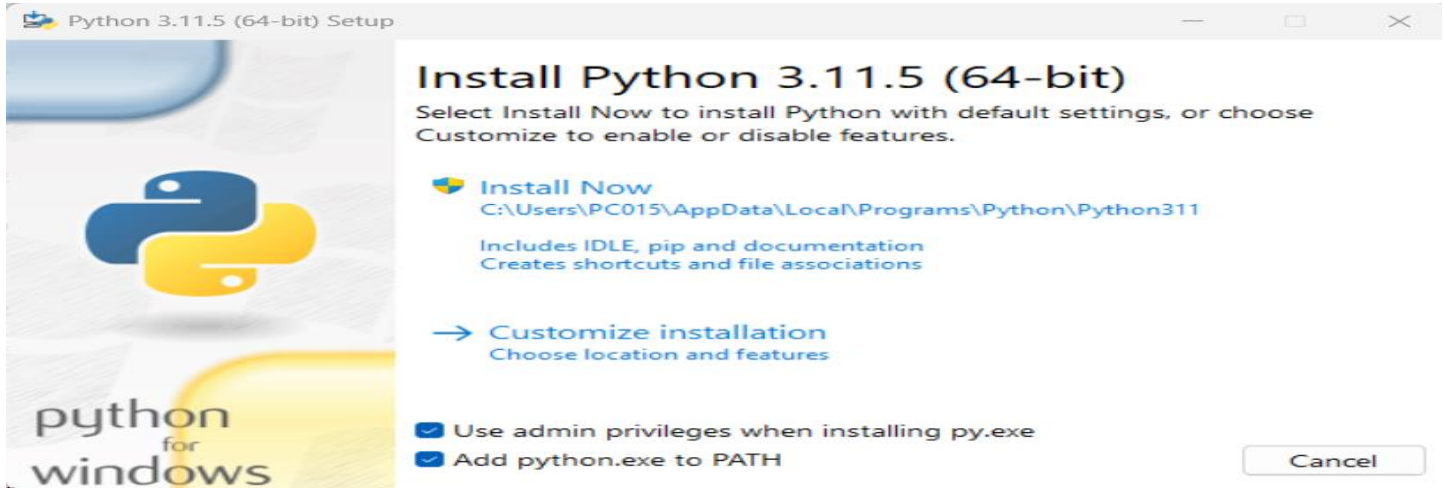
Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.11.5	Aug. 24, 2023	Download	Release Notes
Python 3.10.13	Aug. 24, 2023	Download	Release Notes
Python 3.9.18	Aug. 24, 2023	Download	Release Notes
Python 3.8.18	Aug. 24, 2023	Download	Release Notes
Python 3.10.12	June 6, 2023	Download	Release Notes
Python 3.11.4	June 6, 2023	Download	Release Notes
Python 3.7.17	June 6, 2023	Download	Release Notes

Installing Python on widows..

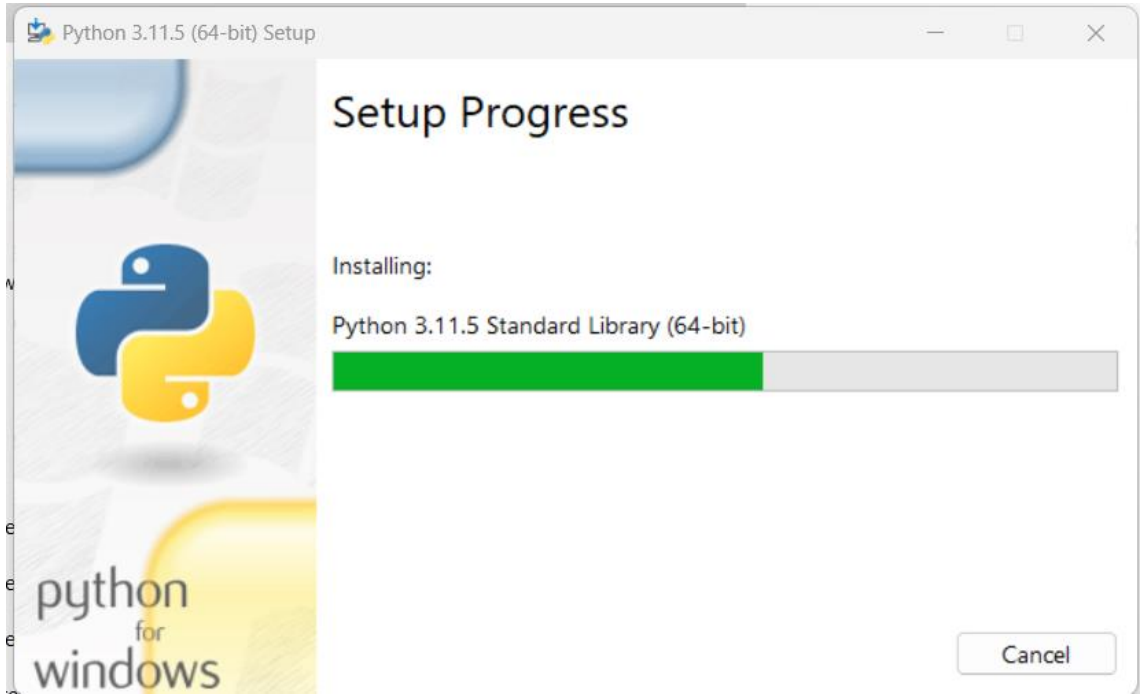
- After the download is complete, double click on the downloaded software to install and the below screen will appear



- Check the “Use admin privileges when installing py.ex” box
- Again check the “Add python.exe to PATH”
- Click install now

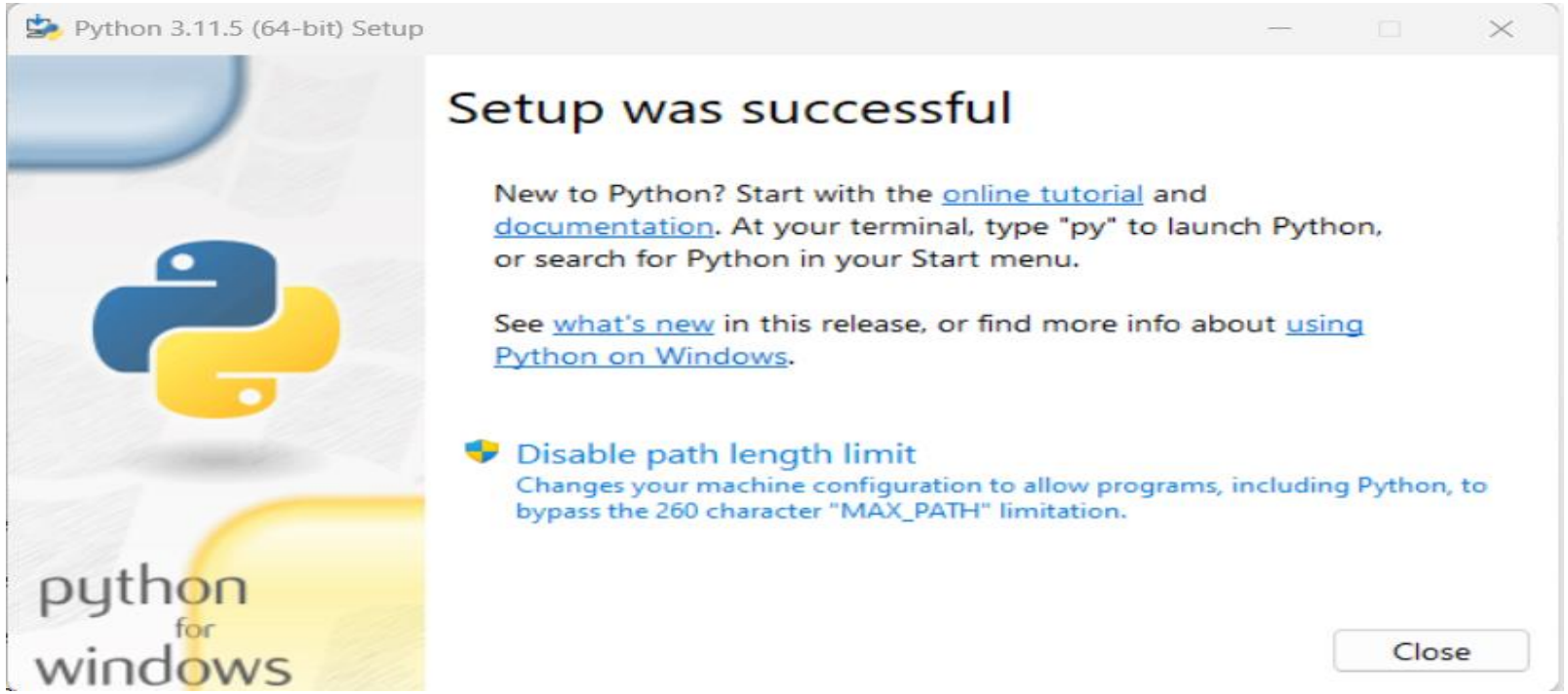
Installing Python on widows...

- Now your installation is in progress. All you do now is to waiting till its complete.



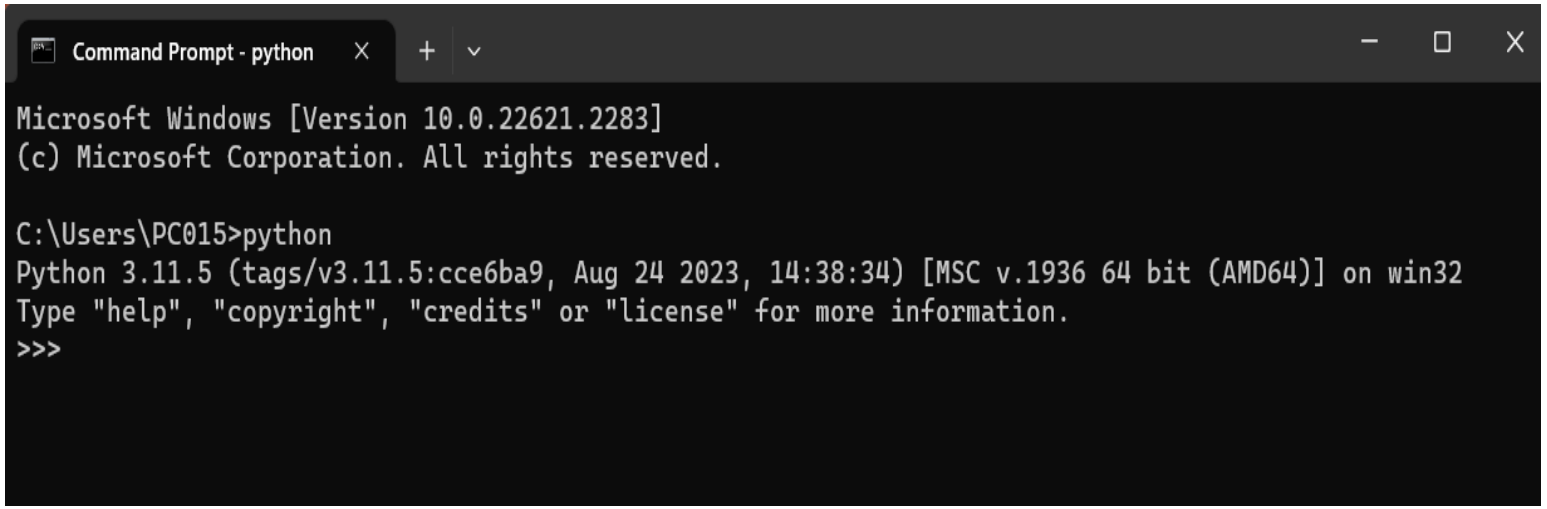
Installing Python on widows....

- Now your installation is complete.
- You may now close or proceed to the python online tutorials or documentation



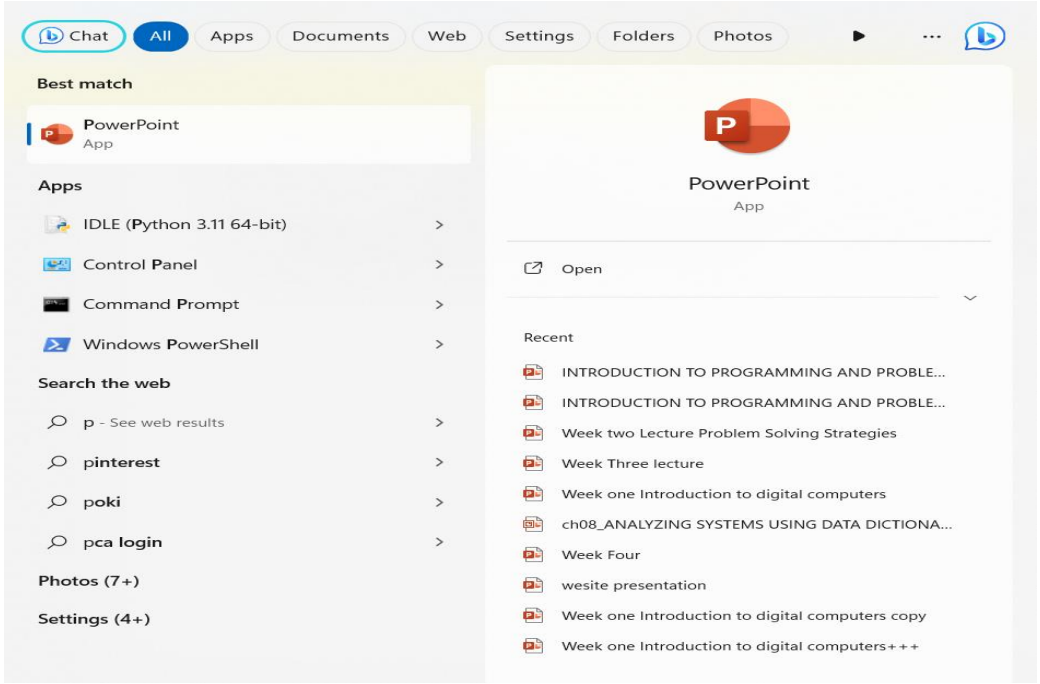
Installing Python on widows.....

- To confirm whether your installation was successfully, open command prompt and type python. The below screen will appear
- You should now be able to see your installed python version and that should be fine for now.



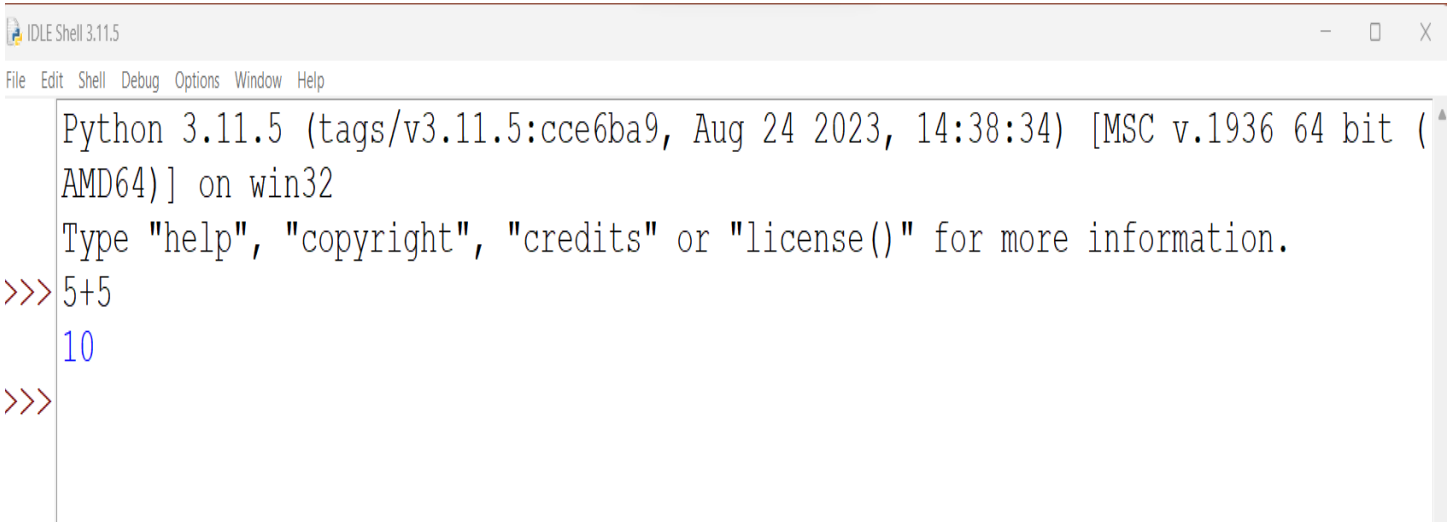
Writing our first program

- Let us now open python and write our first code.
- On the search in your window machine, type python and you should be able to see the below screen.



- Now double click on IDLE(Python 3.11.64-bit)

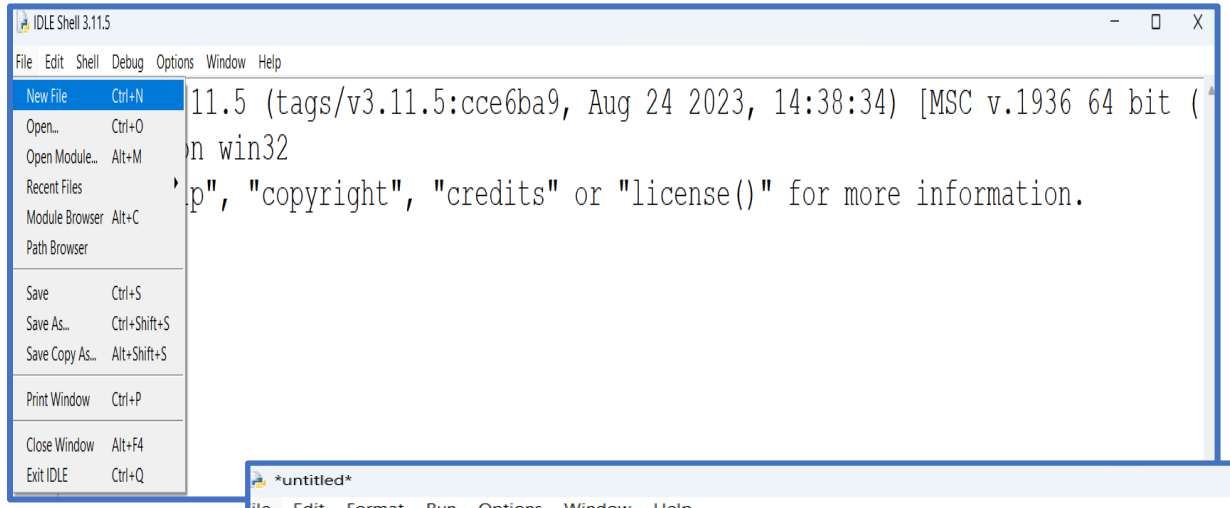
- Your python shell is now open which can act as a calculator .
- As seen below, I added 5+5 and when I pressed enter, I got an output of 10.
- You can also type a string on the shell and get an output on pressing the enter key

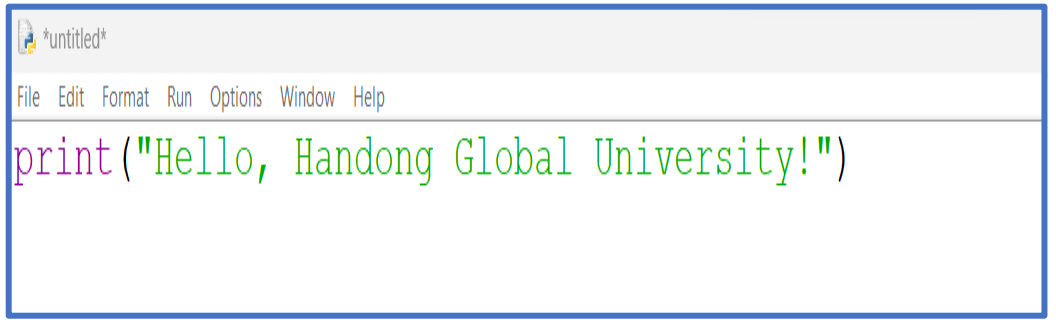


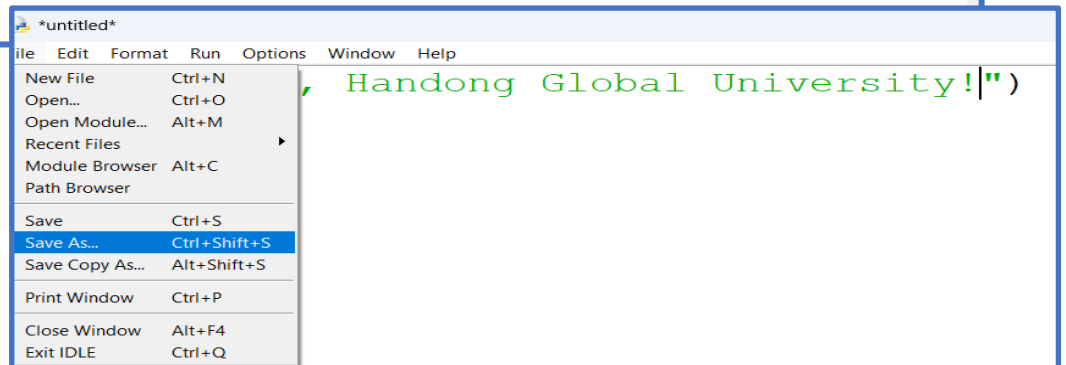
Using a script file (Script Mode Programming)

- The interpreter prompt is best to run a single-line statements of the code
- However, we cannot write the code every-time on the terminal. It is not suitable to write multiple lines of code
- But we can address this by creating a file from the python interactive mode.

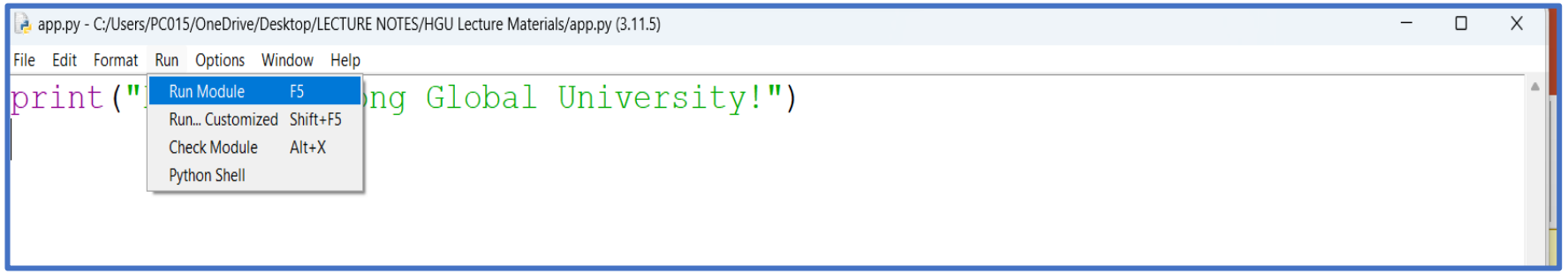
- Click on the file on the top left side of the interactive mode and select new file and that will create a blank file
- Type your python code and save it with a .py, the .py is the extension for a python file.



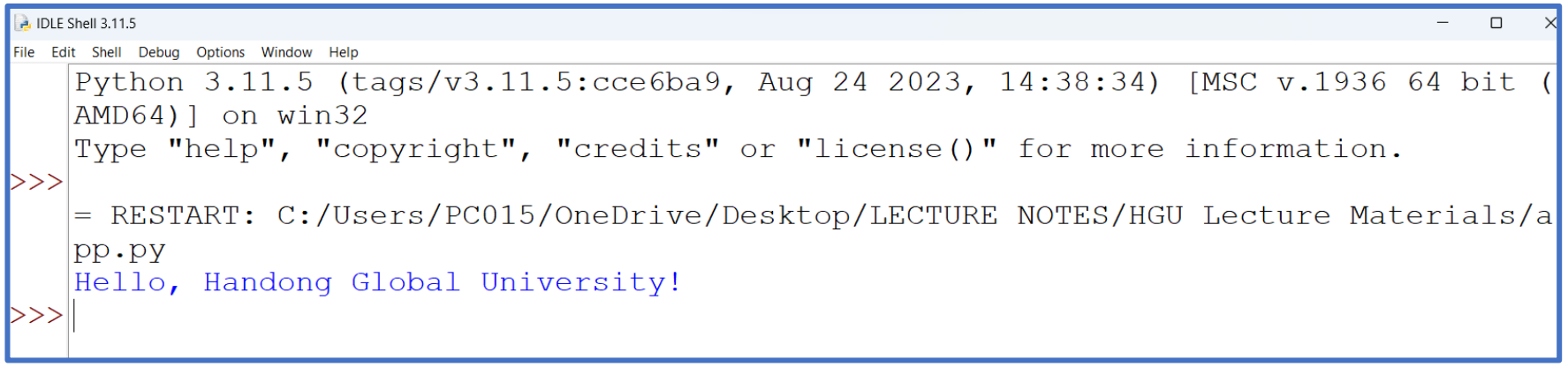




- After your file has been saved, click run or F5

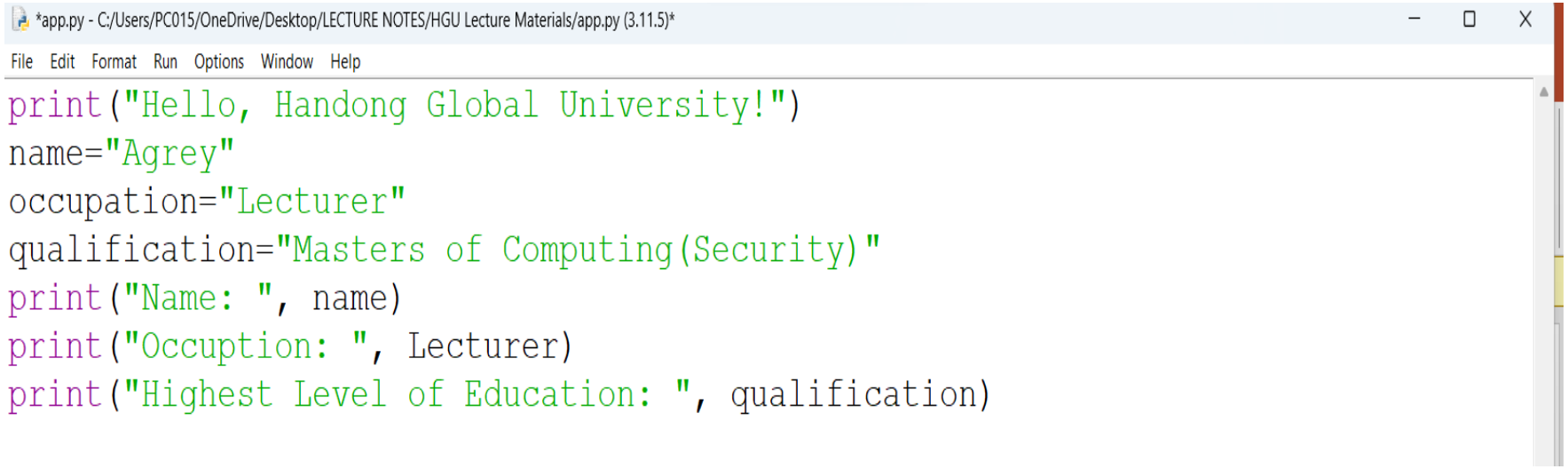


- This will display your output on the idle shell or python interactive mode



Multiple lines

- The script- mode helps us to write multiple lines of code and then run later as can be seen in the figure below.



Internal Working of Python

- **Lexical Analysis (Tokenization)**: When you run a Python program, the source code is read and analyzed by the Python interpreter.
- Lexical analysis involves breaking the source code into individual tokens. Tokens are the smallest units of a program, such as keywords, identifiers, operators, and literals.
- **Parsing (Syntax Analysis)**: The Python interpreter parses the tokenized code to determine its syntax and structure.
- During parsing, the interpreter builds an Abstract Syntax Tree (AST), which represents the hierarchical structure of the program.

Internal Working of Python.

- **Compilation to Bytecode**: After checking the source code, it is then compiled into a bytecode.
- The bytecode is the low level equivalent of the high level code that is platform neutral and executable by the Python Virtual Machine (PVM).
- **Execution**: The bytecode is executed by the Python Virtual Machine (PVM).
- The role of the PVM is to manage the execution of the program, including handling memory, data, and control flow.

Python Implementations

- Cpython, is the default and traditional implementation of Python. The original CPython interpreter was written in the C programming language. It is widely used and forms the basis for the reference implementation of Python.
- Jython, this is an implementation of Python that allows for seamless integration with the Java programming language.
- IronPython, is the .NET implementation of the Python language. It is designed to run on the Microsoft .NET Framework and the .NET Core runtime. One of IronPython's key features is its ability to easily interoperate with other .NET languages such as C# and VB.NET.

Python Implementations+

- Stack less Python, Stack less Python is an extension of CPython (the default Python implementation) that adds support for micro threads and lightweight task let-based concurrency. This allows for efficient multitasking and concurrency in Python applications.
- PyPy, is an alternative Python implementation that focuses on speed and just-in-time (JIT) compilation. Unlike CPython, which uses a bytecode interpreter, PyPy uses a Just-In-Time compiler to execute Python code. This results in significantly improved performance for many Python applications.

Python Basic Syntax

Indentation and Comment in Python

- Indentation is the most significant concept of the Python programming language. Improper use of indentation will end up in **"Indentation Error"** in our code.
- Indentation is nothing but adding whitespaces before the statement when it is needed. Without indentation Python doesn't know which statement to be executed next.
- Indentation also defines which statements belong to which block. If there is no indentation or improper indentation, it will display "**Indentation Error"** and interrupt our code.

Indentation and Comment in Python

- Indentation in Python helps to define which statement belongs to which group.

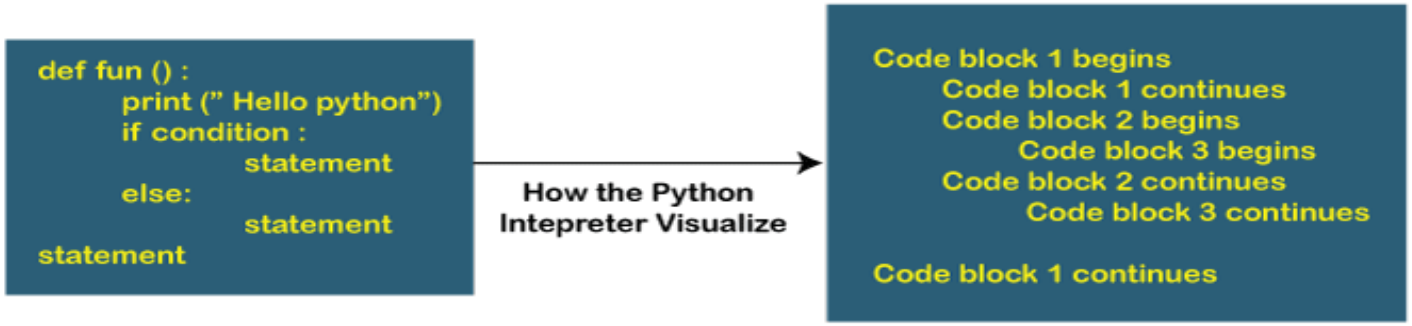


Figure 1: Python indentation[5]

- Statements that are the same level to the right belong to the same block.
- We can use four whitespaces to define indentation

Example of indentation

```
lst = [45, 52, 73, 84, 96]
for x in lst:
    print(x)
    if x==84:
        break
print("End of for loop")
```

Explanation:

- In the above code, the for loop has a code block and the if statement has its own code block inside the for loop. Both indented with four whitespaces. The last **print()** statement is not indented; that means it doesn't belong to for loop.

Comment

- Comments are for us and future developers to understand the thinking behind our code.
- It is often said that on the day we write our code, only God and we are the ones who know about it. But when we come back the following day, it will be only God who knows.
- Therefore, it is important to include these comments for our future selves.
- Comments can also be helpful during debugging;
- We can use them to deactivate parts of the code to assist in locating bugs more easily.
- The '#' symbol is used to indicate a comment.

Example of comments

#The below variable holds the string “Dau”

name=”Dau”

#print(“Print notworking”);

#the above print function is not going to be executed because it has been commented out.

It is important to comment some of the most important parts of the code.

Types of Comment

The two types of comments that exist in Python includes single-line and multi-line comments.

Single-Line Comment - Single-Line comment starts with the hash # character followed by text for further explanation.

```
# the capital of DRC formerly Zaire
city = ”Kinshasa”
print(city)
```

Types of Comment.

Multi-Line Comments – Multiple-lines in Python can be commented out using triple quotation marks as can be seen below.

```
"""

this is a multiline comment

Triple quotation marks are legal

"""
```

PYTHON CHARACTER SET

- The Python character set is the set of all characters that can be used in Python code. It includes all ASCII characters, as well as all Unicode characters. This means that Python can be used to write code in any language, including those that use non-Latin alphabets.

The following are some of the python character sets [1]

- Letters: Upper case and lower case letters
- Digits: 0,1,2,3,4,5,6,7,8,9
- Special Symbols: Underscore (_), (,), [,], {,}, +, -, *, &, ^, %, $, #, !, Single quote(‘), Double quotes(“),
- Back slash(\), Colon(:), and Semi Colon (;)
- White Spaces: (‘\t\n\x0b\x0c\r’), Space, Tab

TOKEN

- A token in Python is the smallest individual unit in a Python program. Tokens are used to build statements and instructions in Python.

There are five types of tokens in Python

- **Delimiter**, delimiters are symbols that perform a special role in Python like grouping, punctuation and assignment. Python uses the following symbols and symbol combinations as delimiters. [1]
- ( ) [ ] { } ,
- : . ‘ = ;
- += -= *= /= //= %=
- &= |= ^= >>= <<= **=

Identifiers

- Identifier is a name given to a variable, function, module, class or other objects. [2]

The following are the rules for naming identifiers in python

- The first character of the identifier must be an alphabet or underscore (_).
- No special character is permissible in identifiers except underscore (_).
- Notwithstanding the above, identifiers must use alphanumeric (a-z, A-Z, 0-9)
- Keywords(language reserved words) must not be used as identifier names
- Identifier names are case sensitive eg. Name is different from NAME or Name.

Tokens+++

- **Literals**, these are values(numbers, string or character) that appears direct in the program e.g.

12=> integer literal,

“Hello”=>String literal

“x” =>character literals

25.5 => floating point literals

[12, 78, 99]=>List literals

Tokens++++

Literals+

{“name”: “Lemi Agrey”, “Occupation”:”Lecturer”}=>Dictionary literal

- Literals can stand alone or used to initialized variables
- We can used the type of function to know the type of a literal value
- For example:
- print(type('x'))
- <class 'str'>

Token++++

- **Keywords** are the words that are having special meaning to the language. They are the language reserved words. To know the different python keywords we can used the following lines of code to print them

```
import keyword

print(keyword.kwlist)
```

Output

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

TOKEN ++

Operators: Operators are symbols used to perform operations on values.

Examples of operators include +, -, *, /, and ==.

The figure below shows the token tree

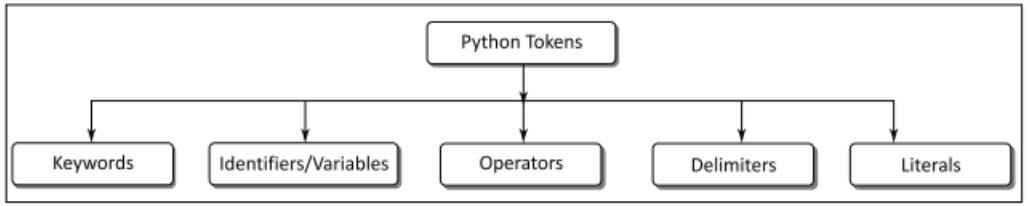


Fig 2: Python Tokens[4]

Token+++++++

- **Variable** is a memory location name or a name that is used to refer to a memory location. A variable is an identifier too in python.
- Unlike other programming languages, python doesn't require an explicit declaration of a data type for a variable. It is smart enough to determine the datatype at run time.
- age=20 is a correct way of declaring and initializing a variable in python however in Java you must begin with the datatype int age=20

Rules for Naming a variable

Just like an identifier the following rules must be taken into accounts while naming a variable

- A variable name must start with either an English letter or underscore (_). first_name, _age, date_of_birth etc.
- A variable name cannot start with a number. E.g. 7num
- Variable can be a group of alphanumeric (a-z, A-Z, 0-9) characters besides the underscore
- Special characters are not allowed in the variable name e.g. lemi@agrey, num!, num.m
- The variable's name is case sensitive. “Deng” and “deng” are two different variables. Use of lower case is the most prepared naming convention.
- Spaces are not allow in a variable name e.g. year of birth will throw an error \
- Keywords can not be used as variable name for example for=20 will raised an invalid syntax error

Variable declaration and Initialization

- The process of creating a variable name is called Variable declaration while the process of assigning a value to a given variable is called variable initialization
- Unlike languages like Java, C, C# etc. that permits a variable to be declared without initializing, in python when we initialized a variable, the interpreter will at the same time treat it as a declaration
- Variable assignment statement format

variable=expression

Initialization

- The equal sign (=) is known as Assignment operator. An expression is any value, text or arithmetic expression, whereas variable is the name of the variable. The value of the expression will be stored in the variable [2]
- Let us look at an example of variable initialization

country="South Sudan"

- The statements above created a memory space/address country and stores value "South Sudan" in it.
- A statement is a line of code or command given to the computer to carry out a particular task 0

Multiword in variable name

- Multiple words can be used as variable names however as mentioned earlier spaces are not permitted in variable. Therefore to name our variable better, the following casings applies
- **Camel Case -** In the camel case, each word or abbreviation in the middle begins with a capital letter. There is no intervention of white space. For example - nameOfStudent, valueOfVariable, etc.

Multiword in variable name

- **Pascal Case -** It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.
- **Snake Case -** In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

➢ Of the three, the most recommended one in python 3 is the snake case however for the purpose of backward compatibility, camel Casing may still be used

Multiple Assignment

- In python, a single value can be assigned to multiple variable however all the variable will be pointing to the same memory address.
- Example

a=b=c=50

- Here all the three variables are sharing the same value 50
- We can also assign multiple value to multiple variables as can be seen below
- x=y=z=20, 89, 10
- In the above statement the value of x will be 20, value of y will be 89 while value of z=10

Types of variable

- In Python, variable scope refers to the region of code where a variable is accessible or visible.
- Python has two main types of variable scope: local scope and global scope.
- Local Scope:
- Variables defined within a function have local scope, meaning they are only accessible within that function.
- These variables are called "local variables," and they are created when the function is called and destroyed when the function exits.
- Attempting to access a local variable outside of the function where it is defined will result in an error.
- Example:

Types of variable++

Code Example: Local Variable

```
def selfish_son():
    local_scope = 10
    print(local_scope)
selfish_son()
print(local_scope)
```

Output

```
10
Traceback (most recent call last):
  File "C:\Users\PC015\OneDrive\Desktop\hello.py", line 5, in <module>
    print(local_scope)
NameError: name 'local_scope' is not defined
```

Types of variable+++

- **Global Scope**
- Variables defined outside of any function or class have global scope, meaning they can be accessed from any part of the code, including inside functions.
- These variables are called "global variables."
- To modify a global variable from within a function, you must use the global keyword to indicate that you are working with the global variable, rather than creating a new local variable with the same name.

Types of variable+++++

```
1. x=89
2. def global_kw():
3.     global x
4.     b=66
5.     print(x)
6.     x=90
7.     print(x)
8. global_kw()
9. print(x)
10. print(b)#This line will cause an error
```

Object reference

- Python is a high level programming language and as such, each of its data items object belongs to a particular class.

```
name="Lam"

print(name)
```

Output: Lam

```
type(name)
```

Output: <class 'str’>

Object reference+

- In the code in our previous slide, we created a string object and printed to the console. We then checked the type of our string object using the Python built-in **type()** function and gave us an output of class str.
- This output simply confirmed that name is holding a value that is of type string
- Python treats variables as symbolic name that is a reference or pointer to an object. They are used to symbolize objects by that particular name
- Consider the example below
- X=89

Object reference++

- In this examples x symbolize an integer object. If we decide to re-assign x to an other variable y, it will be illustrated as follows.
- x=y

- Variable y refers to the same object. This because re-assignment does not create new objects in python.
- If we opted to assign the two variable above to different values, each of them will be pointing to the newly created values

Object Identity

- Each of the python objects have unique identity. There is no room for a situation where more than one objects are the same identifier. Python comes with an inbuilt id() function to identify the identifier of the object. Let us look at the examples below.

```
a=20
b=a
print(id(a))
```

Output: 140705501406600

```
print(id(b))
```

Output: 140705501406600

```
b=50
print(id(b))
```

Output: 140705501407560

Object Identity

- We assigned the **b = a, a** and **b** both point to the same object.
- When we checked by the **id()** function it returned the same number.
- We re-assign **a** to 50; then it referred to the new object identifier.

Datatypes in Python

- Python programs hold data of different data type.
- Python is a dynamically typed language, a reason why we are not usually required to define the datatype while declaring a variable.
- It is the job of the interpreter to bind the variable to the type. For example
- city=‘Juba’
- Although the datatype for variable city is not explicitly defined, the python interpreter automatically defines it implicitly.
- A variable can hold a number of values, for example the city in our above example has a value of string however it’s id is an integer

Datatypes in Python+

Python has the following core datatypes

- Numbers
- Sequence Type
- Boolean
- Set
- Dictionary

Numbers

- Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type.
- The **type()** function is used to print out the datatype of a particular variable.
- Integer is a whole number which can either be positive or negative without a decimal place
- Integers in python are of unlimited length

Example

```
a=30

b=-40

print(a, b)
```

Numbers+

```
print(type(a))

print(type(b))
```

Output

```
30

-40

<class 'int'>

<class 'int'>
```

Numbers++

- Floats, These are numbers which are either positive or negative and have one or more decimal points
- They are also called floating point numbers
- It is accurate up to 15 decimal points.
- a=89.7
- c=22.2365
- d=58.01
- print(a, c, d)

Numbers++

- **Complex -** A complex number contains an ordered pair, i.e., p + q where p and q denote the real and imaginary parts, respectively.
- x2 = complex(3, 56), this function creates a complex number as seen in the out put below

```
print(x2)
```

Output:

```
#This is yet another way of creating complex number
```

Q2=(3+56j)

```
q2=2+5j
```

Sequence type

Sequence types are datatypes that represent an ordered collection that can be manipulated.

There are many sequence type among them are the list, tuple, string, range, Bytes and Byte Arrays,

- Lists is the mutable collection which is defined using square brackets

Example: lst=[25, 78, 10, True, 58.2, “hello”]

- String, is a sequence of immutable characters. This means that; a string can never be changed

Example of a string: str1=“Welcome to Problem Solving”

Sequence type+

- Tuples, tuples are similar to list however they are immutable in nature and they are defined using parenthesis. Example of tuple: tpl=("John", 20, True, 2500000.67)
- Ranges: Ranges are immutable sequences of numbers, often used for interaction in loops. You can create ranges using the range() function.

The range=range(1, 10) #This will print 1, 2, 3, 4, 5, 6, 7, 8, 9

- Bytes and Byte Arrays: Bytes and byte arrays are used to represent sequences of bytes, typically used for binary data. Bytes are immutable, while byte arrays are mutable

wel_byte = b'Welcome to our kingdom'

king_byte_array = bytearray(b'welcome to our kingdom')

Boolean

- Boolean type provides two built-in values, True and False.
- These values are used to determine the given statement true or false.
- It is denoted by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'.
- Consider the following example.

```
# Python program to check the Boolean type
print(type(True))
print(type(False))
print(false)
```

Output:

```
<class 'bool'>
<class 'bool'>
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(type(false))
NameError: name 'false' is not defined. Did you mean: 'False'?
```

Set

- A set is unordered collection of unique elements
- It can be created either using the inbuilt set function set() or can be created using curly braces as seen in the examples below
- Examples:
- first_set=set([56, 12, 89])
- Output: {56, 89, 12}
- second_set={48, 56, 88}

Set+

- print(second_set)
- Output: {48, 56, 88}
- third_set={77, 76, 11, 77, 12, 11}
- print(third_set)

Output: {11, 12, 77, 76}

Our first_set and second_set do not have duplicate however the third set had duplicates but on printing only the unique values were outputted

Dictionary

- A dictionary in Python is a collection of key-value pairs. The keys can be any immutable type, such as strings, numbers, or tuples. The values can be any object type, including other dictionaries.
- Dictionaries are created by enclosing a sequence of key-value pairs in curly braces ({ }). The key-value pairs are separated by commas, and each key-value pair consists of a key, followed by a colon (:), followed by a value.
- For example, the following code creates someone's biodata.

```
1. my_dic={"name": "Alex Towongo", "Occuption":"Church Leader", "age":32, "gender":"male"}
2. print(my_dic)
```

- Output: {'name': 'Alex Towongo', 'Occuption': 'Church Leader', 'age': 32, 'gender': 'male'}

THE print() FUNCTION

- The print function is a function that is used to output to the console
- The print function can take more than one argument. By default a print function has the next line , meaning that if you have two print statements both will be printed on separated lines however this behavior can be overridden by passing in an end keyword and assigning an empty string to.
- The print function can accept a value of any datatype as an argument
- The print function can print out text, variables, statements etc.
- Now let us look at some use cases of the print function

The Print Function

```
print("Hello Kumi University")#printing a string literal
```

Output: Hello Kumi University

Now let us see how to print variables. Let us start by creating some variables

Examples

```
name="Nyero"

description="Old stone age paintings"

location="Nyero Sub County"

size="250 sq-m"
```

The Print Function

```
print("Name of the rock: ", name)

Print("Description of the site: ", description)

print(“Location: ", location)

print("Size: ", size)
```

Output

Name of the rock: Nyero

Description of the site: Old stone age paintings

Location: Nyero

Size: 250 sq-m

The Print Function….

- **Explanation**

In the above slide, we created four variables and wrote four print statements for each of the variables. From the output, we noticed that each of the print statement printed on separate lines, this confirms the assertion that, the print function has a line break argument by default.

Used of the end keyword in the print function

- Now let us see how to override some of the default behaviors of the print function

Let us declare some variables and try to print the values to the console on one line using multiple print statements

```
city1="Juba"

city2="Kampala"

city3="Busan"

city4="Nairobi"

city5="Kinshasa"
```

Use of the end keyword in the print function.

```
print(city1, "is found in South Sudan.", end=" ")

print(city2, "is found in Uganda.", end=" ")

print(city3, "is found in South Korea.", end=" ")

print(city4, "is found in Kenya.", end=" ")

print(city5, "is found in DRC.")
```

Output:

Juba is found in South Sudan. Kampala is found in Uganda. Busan is found in South Korea. Nairobi is found in Kenya. Kinshasha is found in DRC.

Explanation: the above output confirmed that the default behavior of the print function of line breaking can be overwritten by the use of the end keyword.

The input function

- The input function is a python function that is used to take input from the user
- It takes a prompt as an argument however it can also be called without and argument
- The input to print functions are always treated as a string

Example:

- Let us write a variable that ask the user for his name, occupation and level of education and output the results to the consult.

The input function+

- We shall start by firstly creating three variables which we shall then assigned an input functions to each

```
name=input("What is your name? \n")

edu_level=input("What is your highest level of education \n")

occuption=input("What is your occupation \n")

print("Hello ", name, "Your ", edu_level, " is worth its salt, you are doing extremely well as a", occuption)
```

The input function++

Output:

What is your name?

Diing Diing Diing

What is your highest level of education

Msc. Surgery

What is your occupation

Surgeon

Hello Diing Diing Diing Your Msc. Surgery is worth its salt, you are doing extremely well as a Surgeon

Casting the value of input function

- Recall that the input function treats each value it received as a string.
- This behavior makes it difficult to carry our arithmetic operations on the values received by the input function
- To address this problem, we passed the input value to the python inbuilt functions such as int(), float etc to convert the value to the datatype of our interest.
- Note that this conversion only applies to inputs that are numbers

Let us consider the following example

Casting the value of input function+

- Example
- a=int(input("Please enter the value of A: "))
- b=int(input("Please enter the value of B: "))
- c=a+b
- print("The sum of A + B = "+str(c))

Casting the value of input function++

Output

- Please enter the value of A: 56
- Please enter the value of B: 27
- The sum of A + B = 83

Explanation

Our program above passed the input function into an int() in order to direct the user inputs into integers for easy calculation and sum up the two inputs A and B and assigned the results to variable C. It then converts the final results again into string using the str() for the purpose of concatenating it to the string in the print function and printed the output to the console

WRITING SIMPLE PROGRAMS IN PYTHON

- How can we write a program in python to calculate a simple interest
- As we well know, a program is written in step by step manner. Let us consider the following steps
- Step 1: We need to design an algorithm for our above calculator program
- An algorithm gives a description of how the problem is going to be solved and discusses the series actions that need to be taken and the steps that must be followed. This helps the developer to plan and address the problem correctly.
- Step 2: This step involves translating the algorithm into an action

WRITING SIMPLE PROGRAMS IN PYTHON

- Now let us write the algorithm for a simple interest calculation
- Get the principle, interest rate and the time from the user

  si=(p*t*r)/100#formula for calculating simple interest

- Finally display the results to the console
- Now let us write the code

```
1. p=int(input("Please enter the principle amount"))
2. r=int(input("Please enter the interest rate"))
3. t=int(input("Please enter the the repayment time period"))
4. si=(p*r*t)/100
5. print("The simple interest is ", si)
```

WRITING SIMPLE PROGRAMS IN PYTHON +

Output:

Please enter the principle amount 200000

Please enter the interest rate 5

Please enter the repayment time period 5

The simple interest is 50000.0

THE eval() FUNCTION

- Eval is a function which is used to evaluate string as a Python expression. For example if we passed the “Hello, Southern Asia” string to the eval function, it will evaluate and produce the output “Hello, Southern Asia”
- eval(print(“Hello, Southern Asia”))
- We earlier noted that the input function treats every value passed to it as a string and that in order to carry out an arithmetic operation, we must first convert the string value to either float or integer however the use of eval helps us to avoid specifying the exact type we want to convert to as the eval will complete the evaluation of the type for us.
- Amount=eval(input(“Enter amount”))
- The above code will be evaluated and the type will be determine by the value the user enters

A simple Bill Calculator

1. print("Enter the list of items: ")
2. name=input("Enter the Customer name: ")
3. itm1=input("Enter the first item: ")
4. qty1=eval(input("Enter the quantity: "))
5. p1=eval(input("Please enter the unit price: "))
6. itm2=input("Enter the second item: ")
7. qty2=eval(input("Enter the quantity2: "))

A simple Bill Calculator+

```
8)  p2=eval(input("Please enter the unit price"))
9)  amt=(qty1*p1)
10) amt1=(qty2*p2)
11) vat=(amt+amt1)*0.18
12) total=amt+amt1+vat
13) print("=========================================")
14) print("Customer name",name)
15) print("01.",itm1,"              ", amt)
16) print("02.",itm2,"              ", amt1)
17) print("03. Value Added Tax(VAT)",vat)
18) print("Total amout spent:      ", total)
```

A simple Bill Calculator++

- Enter the bills details
- Enter the Customer name: Lemi Agrey Oliver
- Enter the first item: Beer
- Enter the quantity: 10
- Please enter the unit price: 3000
- Enter the second item: Pizza
- Enter the quantity: 24
- Please enter the unit price: 30000
- ==========

A simple Bill Calculator++

Customer name Lemi Agrey Oliver

01. Beer 30000

02. Pizza 120000

03. Value Added Tax(VAT) 27000.0

Total amout spent: 177000.0

FORMATTING NUMBER AND STRINGS

- Formatting makes strings looks more presentable
- Format() is used to format string in python
- Using the format function in numbers can help us increased or decreased the number of decimal places
- format(item, format-specifier)
- Print(33.23909, format(33, '.2f'))
- Output: 33.00

FORMATTING NUMBER AND STRINGS +

Examples

- r = int(input(‘Enter Radius: ‘))
- print(‘ Radius = ‘, r)
- pi = 3.1428
- a= pi * r * r
- print(a)#28.2852
- print(‘ Area= ‘,format(a,’.2f’))
- Output
- Enter Radius: 3
- Radius = 3
- Area=28.28

Table showing the frequently used format specifiers

Specifier	*Format*
10.2f	Format floating point number with precision 2 and width 10.
<10.2f	Left Justify the floating point number.
>10.2f	Right Justify the formatted item.
10X	Format integer in hexadecimal with width 10
20s	Format String with width 20
10.2%	Format the number in decimal

Fig2. Format specifier table[3]

The ord and chr Functions

- Since the computer understands only the binary language, it then follows that all computer characters are stored in binary form
- The process of mapping a character to its binary equivalent is called character encoding.
- There are many encoding schemes, among them is the ASCII(America Standard for code information interchange.
- ASCII is a 7 bits encoding that is used for representating alphanumeric characters and punctuation marks.
- The ASCII character range is from 0-127

The ord and chr Functions

- To find out the ASCII equivalent of character in python we used the ord built in function and to convert an ASCII number to character we used chr()
- print(ord('p'))
- 112
- print(ord('q'))
- 113
- print(chr(113))
- q
- print(chr(65))
- A

Built in functions

- Other common used print and input function, Python provides a wide range of built-in functions that are readily available for use without the need to import additional modules or libraries
- The following are some of the examples
- int(), Converts a value to an integer.
- float(), Converts a value to a floating-point number.
- str(), Converts a value to a string.
- bool(), Converts a value to a Boolean (True or False).
- abs() For getting the absolute value

Summary

- In this chapter, we learned about the basics of Python programming.
- We started by introducing ourselves to Python programming, discussed a brief history of the Python language, installed the Python interpreter, and wrote our first piece of code.
- We then learned about Python character sets and provided some examples.
- We went further to discuss tokens and broke them down into delimiters, operators, variables, etc.
- Additionally, we learned about the **print()** function, which is used for displaying data on the console, and then learned about the **input()** function, which accepts inputs from the user.
- We also discussed the Python built-in function **eval()**. Finally, we concluded by covering number and string formatting

References

[1] *Introduction Programming and problem solving with python*, Ashok N. Kamthane, Amit A. Kamthane, McGraw Hill Education (India) Private Limited,2018, page 24

[2] *Introduction to computing and problem-solving using python*, E Balagurusamy, McGraw Hill Education (India) Private Limited, 2016, page 43

[3] *Introduction Programming and problem solving with python*, Ashok N. Kamthane, Amit A. Kamthane, McGraw Hill Education (India) Private Limited,2018, page 47

[4] *Introduction Programming and problem solving with python*, Ashok N. Kamthane, Amit A. Kamthane, McGraw Hill Education (India) Private Limited,2018, page 24

[5] *Indentation error in Python - Javatpoint*. www.javatpoint.com. (n.d.). https://www.javatpoint.com/indentation-error-in-python