LETS DO SOME SIMPLE CODE IN PYTHON

The Python Command Line

- To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

```
C:\Users\Your Name>python
```

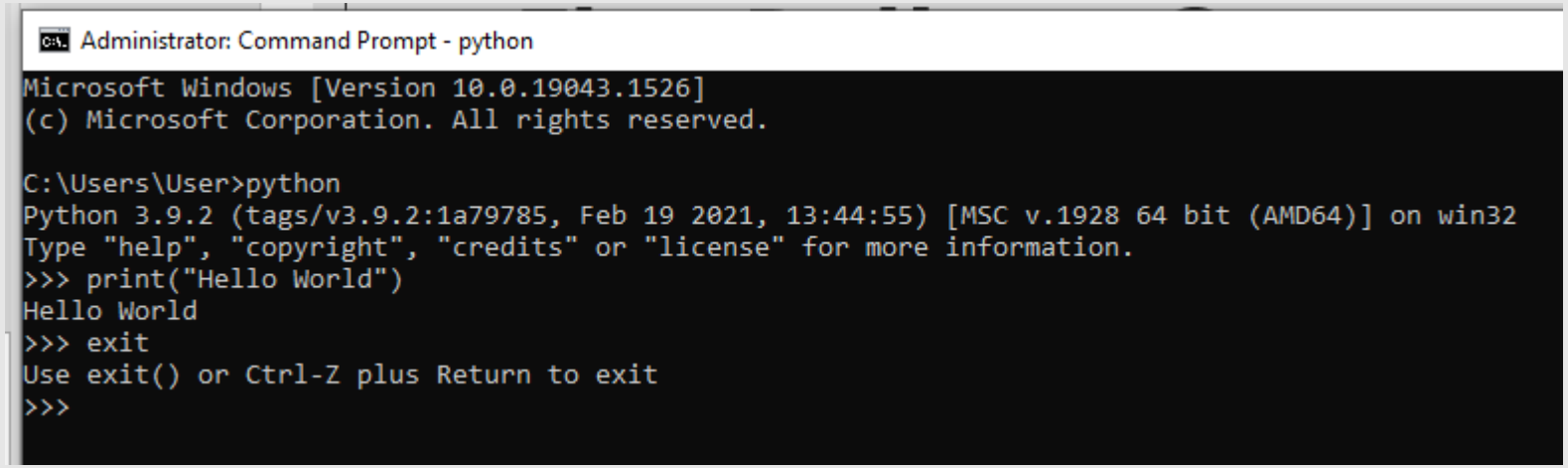
- Or, if the “python” command did not work, you can try “py”:

```
C:\Users\Your Name>py
```

- Whenever you are done in the python command line, you can simple type the following to quit the python command line interface.

```
exit()
```

Example:



- Python syntax can be executed by writing directly in the Command Line:

  >>>print(“Hello World”)

  Hello, World

- Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

  C:\Users\Your Name>python myfile.py

Python Indentation

- **Indentation** refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is **vey important.**
- Python uses indentation **to indicate a block of code.**
- The number of spaces is up to you as a programmer, but it has to be at least one.
- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error.

Python Comments

- **Comments** can be used to explain Python code.
- Can be used to make the code more readable.
- Can be used to prevent execution when testing code.
- Starts with a **#**, and Python will ignore them:

```
#This is a comment
print(“Hello, World!”)
print(“Hello, Python!”) #This is a comment
#print(“Hello, World lagi!”)
print(“Assalamualaikum”)
```

Python Comments (cont.)

- Python does not really have a syntax for multi line comments.
- To add a multiline comment you could insert a # for each line.

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

- Or not quite as intended, you can use a multiline string.
- Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (**triple quotes**) in your code, and place your comment inside it:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Python Variables

```
Administrator: Command Prompt - python
Microsoft Windows [Version 10.0.19043.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>python
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> name="Zarina"
>>> age=23
>>> print(name, age)
Zarina 23
>>>
```

- **Variables** are containers for storing data values.
- In Python, variables are created when you assign a value to it:
- A variable is created the moment you first assign a value to it.
- Variables do not need to be declared with any particular type, and can even change type after they have been set.

```
Administrator: Command Prompt - python
Microsoft Windows [Version 10.0.19043.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>python
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 4 # x is of type int
>>> x = "Sally" # x is now of type str
>>> print(x)
Sally
>>>
```

Casting

- If you want to specify the data type of variable, this can be done with casting.

```
x = str(3) # x will be ‘3’
y = int(3) # y will be 3
z = float(3) # z will be 3.0
```

Get the Type

- You can get the data type of a variable with the type() function.

```
x = 5
y = “John”
print(type(x))
print(type(y))
```

Single or Double Quotes

- **String** variables can be declared either by using single or double quotes:

```
x = “John”
# is the same as
x = ‘John’
```

Python code : Case Sensitive

- Variable names are case-sensitive.

```
a = 4
A = “Sally”
# A will not overwrite a
```

Python – Variable Names

- A variable can have a short name (like x and y) or more descriptive name (age, carname, total_volume). Rules for Python variables name:

must start with a letter or the underscore character

CANNOT start with a number

can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

are **CASE-SENSITIVE** (age, Age and AGE are three different variables)

Python – Variable Names (cont)

- Legal variable names:

  myvar = “John”

  my_var = “John”

  _my_var = “John”

  myVar = “John”

  MYVAR = “John”

  myvar2 = “John”

- Illegal variable names:

  2myvar = “John”

  my-var = “John”

  my var = “John”

Remember that variable names are case-sensitive.

Multi Words Variable Names

- Variable names with more than one word can be difficult to read. There are several techniques you can use to make them more readable:
- Camel Case
  - Each word, except the first starts with a capital letter:

    myVariableName = “John”
- Pascal Case
  - Each word starts with a capital letter:

    MyVariableName = “John”
- Snake Case
  - Each word is separated by an underscore character:

    my_variable_name = “John”

Output Variables (+ symbol)

- The Python print statement is often used to output variables.
- To combine both text and a variable, Python uses the + character:

```
x = “awesome”
print (“Python is ” + x)
```

- You can also use the + character to add a variable to another variable:

```
x = “Python is ”
y = “awesome”
z = x + y
print (z)
```

Output Variables (+ symbol) (cont)

- For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print (x + y)
```

- If you try to combine a string and a number, Python will gave you an error:

```
x = 5
y = “John”
print (x + y)
```

CHAPTER 1:
INTRODUCTION TO
BASIC OPERATIONS IN PYTHON

Lesson Learning Outcome:

1

Explain literals in Python

a. Integers
b. Floats
c. Strings
d. Boolean

2

Explain operators in Python

a. Operators and expression
b. Arithmetic operators

3

Follow rules in assigning variables for Python

a. Naming variables
b. Assigning variables
c. Python keywords

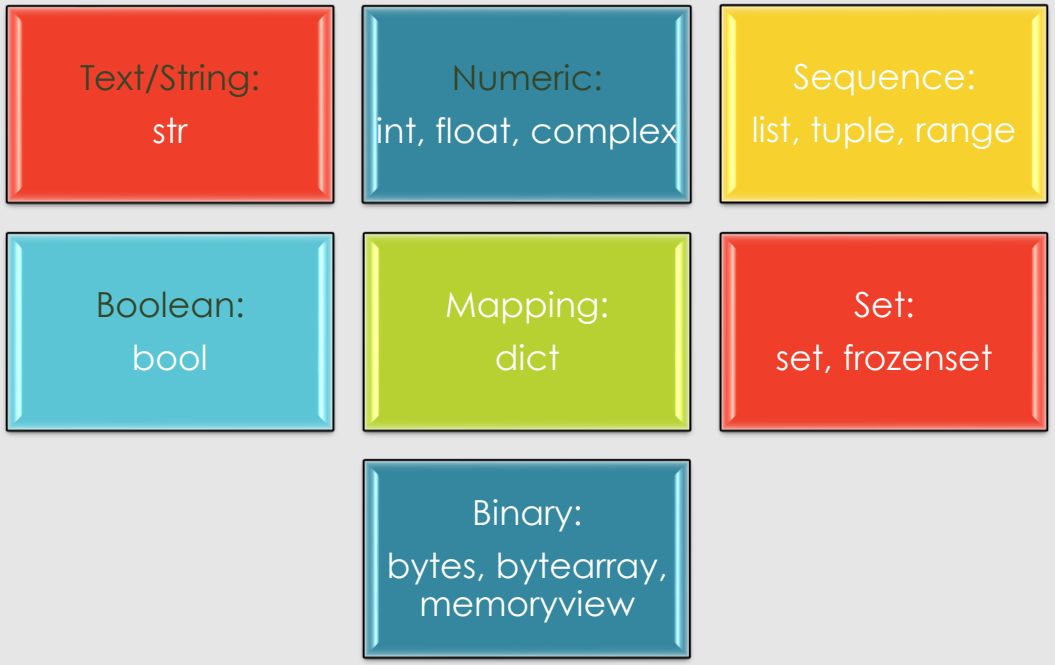
4

Construct Python literals

a. Input ( ) function
b. String operator
c. Strings into numbers
d. Numbers into strings

Types of Python Literals

- Built-in Data Types
- In programming, data type is an important concept.
- Variables can store data of different types and different types can do different things.
- Python has the following data types built-in by default in these categories:
- You ca get the data type of any object by using the type( ) function:



Python Operators

- **Operators** are used **to perform operations on variables and values.**
- In the example below, we use the **+** operator to add together two values:
  - Print(10 + 5)
- Python divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Logical operators
  - Identity operators
  - Membership operators
  - Bitwise operators

Python Arithmetic Operators

- Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Example

```
x = 15
y = 4
print('x + y =', x+y)
print('x - y =', x-y)
print('x * y =', x*y)
print('x / y =', x/y)
print('x // y =', x//y)
print('x ** y =', x**y)
```

Output:

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625

***Repl Closed***
```

Python Assignment Operators

- Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

- Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

- Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Python Membership Operators

- Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

- Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Python Keywords

- Python has a set of keywords that are reserved words that CANNOT be used as variable names, function names, or any other identifiers:

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

Python *input()* Function

- Definition and Usage
  - The input() function allows user input
- Syntax
  - Input(*prompt*)
- Example 1:

```
print ('Enter your name:’)
x = input ()
print('Hello, ' + x)
```

- Example 2:
  - Use the prompt parameter to write a message before the input:

```
x = input('Enter your name:')
print('Hello, ' + x)
```

```
C:\Users\My Name>python demo_input2.py
Enter your name:sharizan
Hello, sharizan
```

String Operators in Python

- String operators represent the different types of operations that can be employed on the program's string type of variables.
- Python allows several string operators that can be applied on the python string are as below:

Assignment operator	"="
Concatenate operator	"+"
String repetition operator	"*"
String slicing operator	"[]"
String comparison operator	"==" & "!="
Membership operator	"in" & "not in"
Escape sequence operator	"\"
String formatting operator	"%" & "{}"

String Formatting Operator “%”

- String formatting operator is used to format as string as per requirement.
- To insert another type of variable along with string, the “%” operator is used along with python string. “
- %” is prefixed to another character indicating the type of value we want to insert along with the python string.
- Please refer to the table for some of the commonly used different string formatting specifiers:

Operator	Description
%d	Signed decimal integer
%u	Unsigned decimal integer
%c	Character
%s	String
%f	Floating-point real number

Example:

```
name = “Faris"
age = 19
marks = 95

string1 = 'Hey %s' % (name)
print(string1)

string2 = 'my age is %d' % (age)
print(string2)

string3= 'Hey %s, my age is %d' % (name, age)
print(string3)

string4= 'Hey %s, my subject mark is %f' % (name, marks)
print(string4)
```

Output:

```
Hey Faris
My age is 19
Hey Faris, my age is 19
Hey Faris, my subject mark is 95.000000

***Repl Closed***
```

Manipulate Python Literals Using Converting Operation

- Python allows you to convert strings, integers and floats interchangeably in a few different ways. The simplest way to do this is using the basic **str(), int()** and **float()** functions.

```
#simple calculator add only
no1=input("Enter first number: ")
no2=input("Enter second number: ")

total=no1+no2

print(no1+"+"+no2+"="+total)
```

```
Enter first number: 10
Enter second number: 20
10+20=1020

***Repl Closed***
```

This will caused error. You **CANNOT perform mathematical operations using string.** By default input() method reads a line from the input (usually from the user), converts the line into a **string** by removing the trailing newline, and returns it.

Converting Strings Into Numbers

```
#simple calculator add only
no1=int(input("Enter first number: "))
no2=int(input("Enter second number: "))

total=no1+no2

print (str(no1)+"+"+str(no2)+"="+str(total))
```

Output:

```
Enter first number: 10
Enter second number: 20
10+20=30

***Repl Closed***
```

You need to cast/convert the input to number, and during printing you need to cast/convert number to integer.

If you try to combine a string and a number, Python will give you an error. So does doing mathematical operations with string.

Converting Numeric Into Strings

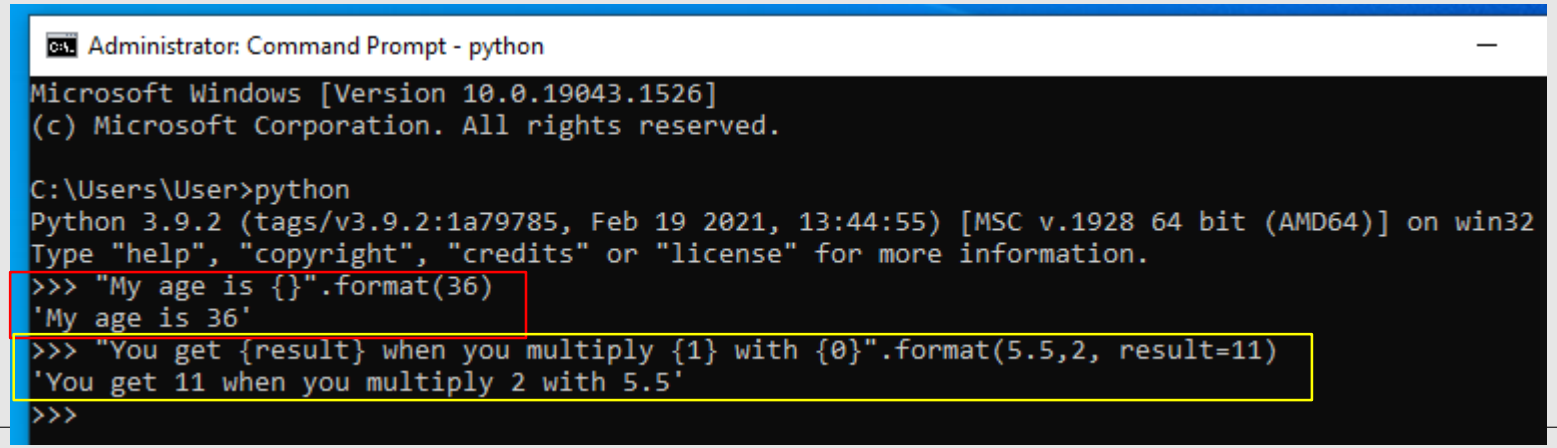
- Using the **str()** Function
- The str() function can be used to change any numeric type to a string.

```
nama="Ahmad Siddiq"
umur=9

print("Salam, nama saya "+nama+". Umur saya "+str(umur)+" tahun")

```

Casting is when you **convert a variable value from one type to another.** This is, in Python done with functions such as int() or float() or str(). A very common pattern is that you convert a number, currently as a string into a proper number.

Using the *format()* Function

- Another way of converting numerics to strings:
  - the **format()** function
    - Allows you to set placeholders within a string and then convert another data type to a string and fill the placeholders.
- To use the function, simply write a string followed by .format() and pass the arguments for the placeholders.
- The arguments in the .format() function can also be referred to individually, using their positions or variable names:



LAB ACTIVITY 1 & 2

See you in NEXT TOPIC