PROTECH GROUP

INTRODUCTION TO PYTHON

ITC C506-3011 ITE ELECTIVE 3

WHAT IS PYTHON ?

- Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

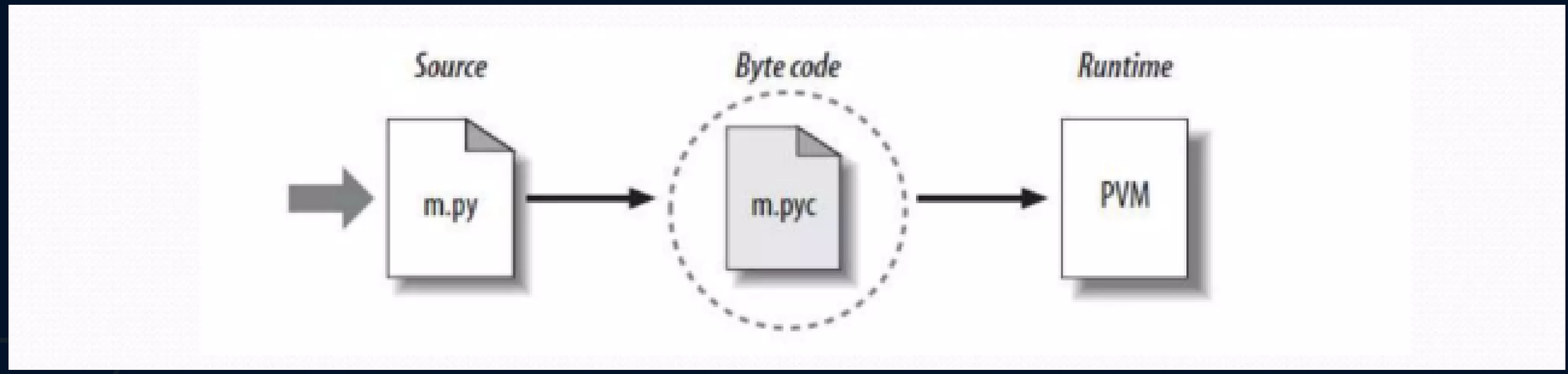
- It is a general purpose programming language that is often applied in scripting language. Python is a programming language as well as scripting language; it is also called as Interpreted language.

IT IS CALLED GENERAL-PURPOSE PROGRAMMING LANGUAGE AS IT IS USED IN ALMOST EVERY DOMAIN WE CAN THINK OF AS MENTIONED BELOW:

- Web Development
- Software Development
- Game Development
- AI & ML
- Data Analytics

PYTHON CODE EXECUTION

Python's traditional runtime execution model: source code you type is trranslated to byte code, which is then run by the Python Virtual Machine. Your code is automatically compiled, but them it is interpreted.



- SOURCE CODE EXTENSION IS .PY
- BYTE CODE IS .PYE (COMPILED PYTHON CODE)

PYTHON DATA TYPES

01

TEXT TYPE

- str

02

NUMERIC TYPES

- int
- complex
- float

03

SEQUENCE TYPES

- list
- range
- tuple

04

MAPPING TYPE

- dict

05

SET TYPES

- set
- frozenset

06

BOOLEAN TYPE

- bool

07

BINARY TYPES

- bytes
- memoryview
- bytearray

08

NONE TYPE

- NoneType

PYTHON ARITHMETIC OPERATORS

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

PYTHON ASSIGNMENT OPERATORS

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

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

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

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

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

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

PYTHON OPERATOR PRECEDENCE

Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

NOTE:

IF TWO OPERATORS HAVE THE SAME PRECEDENCE, THE EXPRESSION IS EVALUATED FROM LEFT TO RIGHT.

PYTHON SYNTAX

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 IDENTATION

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 VERY IMPORTANT. PYTHON USES INDENTATION TO INDICATE A BLOCK OF CODE.

CORRECT

```
if 5 > 2:
  print("Five is greater than two!")
```

WRONG

```
if 5 > 2:
print("Five is greater than two!")
```

NOTE:

- THE NUMBER OF SPACES IS UP TO YOU AS A PROGRAMMER, THE MOST COMMON USE IS FOUR, 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 VARIABLES

VARIABLES

- VARIABLES ARE CONTAINERS FOR STORING DATA VALUES.

CREATING VARIABLES

- PYTHON HAS NO COMMAND FOR DECLARING A VARIABLE. 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.

```
x = 5
y = "John"
print(x)
print(y)
```

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

CASE-SENSITIVE

- VARIABLE NAMES ARE CASE-SENSITIVE.

```
a = 4
A = "Sally"
#A will not overwrite a
```

PYTHON VARIABLES

CASTING

- IF YOU WANT TO SPECIFY THE DATA TYPE OF A 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))
```

```
<class 'int'>
<class 'str'>
```

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 VARIABLES-NAMES

A VARIABLE CAN HAVE A SHORT NAME (LIKE X AND Y) OR A MORE DESCRIPTIVE NAME (AGE, CARNAME, TOTAL_VOLUME). RULES FOR PYTHON VARIABLES:

- A VARIABLE NAME MUST START WITH A LETTER OR THE UNDERSCORE CHARACTER
- A VARIABLE NAME CANNOT START WITH A NUMBER
- A VARIABLE NAME CAN ONLY CONTAIN ALPHA-NUMERIC CHARACTERS AND UNDERSCORES (A-Z, 0-9, AND _)
- VARIABLE NAMES ARE CASE-SENSITIVE (AGE, AGE AND AGE ARE THREE DIFFERENT VARIABLES)

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

LEGAL VARIABLE NAMES

ILLEGAL VARIABLE NAMES

```
2myvar = "John"
my-var = "John"
my var = "John"
```

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"
```

PYTHON - OUTPUT VARIABLES

- THE PYTHON PRINT() FUNCTION IS OFTEN USED TO OUTPUT VARIABLES.

```
x = "Python is awesome"
print(x)
```

```
Python is awesome
```

- IN THE PRINT() FUNCTION, YOU OUTPUT MULTIPLE VARIABLES, SEPARATED BY A COMMA:

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

```
Python is awesome
```

- YOU CAN ALSO USE THE + OPERATOR TO OUTPUT MULTIPLE VARIABLES:

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

```
Python is awesome
```

PYTHON - OUTPUT VARIABLES

- FOR NUMBERS, THE + CHARACTER WORKS AS A MATHEMATICAL OPERATOR.

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

```
15
```

- IN THE PRINT() FUNCTION, WHEN YOU TRY TO COMBINE A STRING AND A NUMBER WITH THE + OPERATOR, PYTHON WILL GIVE YOU AN ERROR.

```
x = 5
y = "John"
print(x + y)
```

ERROR

- THE BEST WAY TO OUTPUT MULTIPLE VARIABLES IN THE PRINT() FUNCTION IS TO SEPARATE THEM WITH COMMAS, WHICH EVEN SUPPORT DIFFERENT DATA TYPES.

```
x = 5
y = "John"
print(x, y)
```

```
5 John
```

PYTHON COMMENTS

- COMMENTS CAN BE USED TO EXPLAIN PYTHON CODE.
- COMMENTS CAN BE USED TO MAKE THE CODE MORE READABLE.
- COMMENTS CAN BE USED TO PREVENT EXECUTION WHEN TESTING CODE.

CREATING COMMENTS

- COMMENTS STARTS WITH A #, AND PYTHON WILL IGNORE THEM.
- COMMENTS CAN BE PLACED AT THE END OF A LINE, AND PYTHON WILL IGNORE THE REST OF THE LINE.
- A COMMENT DOES NOT HAVE TO BE TEXT THAT EXPLAINS THE CODE, IT CAN ALSO BE USED TO PREVENT PYTHON FROM EXECUTING CODE:

```
#This is a comment
print("Hello, World!")
```

```
print("Hello, World!") #This is a comment
```

```
#print("Hello, World!")
print("Cheers, Mate!")
```

PYTHON COMMENTS

MULTILINE COMMENTS

- PYTHON DOES NOT REALLY HAVE A SYNTAX FOR MULTILINE COMMENTS.
- TO ADD A MULTILINE COMMENT YOU COULD INSERT A # FOR EACH LINE.
- 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.
- AS LONG AS THE STRING IS NOT ASSIGNED TO A VARIABLE, PYTHON WILL READ THE CODE, BUT THEN IGNORE IT, AND YOU HAVE MADE A MULTILINE COMMENT.

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

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

PROTECH GROUP

THANK YOU !

MEMBERS:

ERWIN JAMES RUPISAN

CLARK CERVANTES

SAMANTHA MAE MADRONERO

GLEN PEREZ

JOPHET HERNANDEZ

BSIT 3011