ASSIGNMENT 6

20 May
Python Basic - 1

1 What are keywords in python? Using the keyword library, print all the python keywords.

Keywords in Python are reserved words that have special meanings and purposes within the language. These keywords are used to define the structure and behavior of Python programs and cannot be used as identifiers (variable names, function names, etc.).

You can use the keyword module to access a list of all Python keywords.

```
import keyword

python_keywords = keyword.kwlist          # Get the list of Python keywords

for keyword in python_keywords:           # Print all the keywords

   print(keyword)
```

2 What are the rules to create variables in python?

variables are used to store and manipulate data. To create variables in Python, you need to follow some rules:

* Variable names must start with a letter (a-z, A-Z) or an underscore (_).
* after 1 st letter They can be followed by letters, digits (0-9), or underscores.
* Variable names are case-sensitive, so `myVar` and `myvar` are considered different variables.
*cannot use Python keywords (reserved words) as variable names
*Variable names cannot contain spaces
*Variable names cannot contain special characters like!, @, etc.

Here are some examples of valid variable names:

```
name = "John"
age = 30
total_count = 100
my_var = 42
```

3 What are the standards and conventions followed for the nomenclature of variables in python to improve code readability and maintainability?

following consistent naming conventions for variables is essential to improve code readability and maintainability. Python's PEP 8 (Python Enhancement Proposal 8) provides a style guide for writing clean, readable code, including guidelines for variable naming. Here are some of the key standards and conventions for variable naming in Python:

Use Descriptive Names: Variable names should be descriptive and indicative of their purpose. Choose names that make it clear what the variable represents.Avoid single-character variable names (e.g., x, y, i) unless they are used in a well-understood context (e.g., loop counters).

Use Snake Case: For variable names, use snake_case, which means words are separated by underscores (e.g., my_variable_name).Snake case is the preferred naming convention for variables, functions, and module-level variables.

Constants: Constants, such as values that should not be modified, are typically named using ALL_CAPS with underscores (e.g., `MAX_VALUE`, `PI`).

Class Names: Use CamelCase (also known as CapWords or PascalCase) for class names (e.g., MyClass, EmployeeRecord).Class names should follow the CapWords convention, starting with an uppercase letter.

Private Variables: To indicate that a variable is intended for internal use within a module or class and should not be accessed directly outside of it, prefix the variable name with a single underscore (e.g., _internal_var).

- For stronger indication of privacy (name mangling), you can use double underscores as a prefix (e.g., __private_var).

Function Names: Function names should also use snake_case (e.g., calculate_average).

- Function names should be descriptive of their purpose and follow the same conventions as variable names.

Module Names: Module names should be short, all lowercase, and descriptive. Avoid using special characters or spaces (e.g., my_module.py).

Consistency: Maintain consistent naming conventions throughout your codebase. If you choose a naming style, stick with it consistently.

Avoid Reusing Built-in Names: Avoid using names of built-in Python functions, modules, or objects as variable names to prevent conflicts (e.g., don't use names like list, str, or int for your variables).

Abbreviations: Use common abbreviations that are widely understood (e.g.,max for maximum, avg for average).

By following these naming conventions and standards,enhance the readability and maintainability of your Python code. Consistency in naming helps others who read your code understand its purpose and structure more easily.

4 What will happen if a keyword is used as a variable name?

If you use a keyword (reserved word) as a variable name in Python, you will encounter a SyntaxError. Python will raise an error because keywords have predefined meanings and are reserved for specific language constructs. Attempting to use a keyword as a variable name violates Python's syntax rules.

For example, if you try to use the keyword if as a variable name:

```
if = 42          # This will result in a SyntaxError
```

5 For what purpose def keyword is used?

def keyword is used to define functions. Functions are blocks of code that perform a specific task or set of tasks and can be executed or "called" at various points in a program.

```
def function_name(parameters):
   # Function body
   # Code to perform a specific task

   return result                    # Optional return statement
```

6What is the operation of this special character '\'?

the special character '\' is known as the backslash. It is used as an escape character to represent certain special characters or to create escape sequences in strings and characters. The backslash is used to indicate that the character following it should be treated differently, or it has a special meaning. Here are some common uses of the backslash in Python:

Escape Sequences in Strings: The backslash is used to create escape sequences in strings to represent special characters. For example:

- \n represents a newline character.
- \t represents a tab character.
- \'represents a single quote character within a single-quoted string.
- \" represents a double quote character within a double-quoted string.
- \\represents a literal backslash character.

```
print("Hello\nWorld")        # Output: Hello
                             #         World

print('I\'m Python')         # Output: I'm Python
```

Escape Characters: The backslash is used as an escape character in various contexts, including regular expressions and file paths. For example:

- In regular expressions, it's used to escape special characters like . and *
- In file paths, it's used to escape directory separators, such as '\\' on Windows or '/' on Unix-based systems.

```
import re

pattern = re.compile(r'\d+')
match = pattern.search('42 is a number')
```

Line Continuation: The backslash can be used at the end of a line to indicate that the code continues on the next line. This is useful for breaking long lines of code for readability.

```
long_string = "This is a very long string that \
        spans multiple lines for readability."
```

Unicode Escape: In Unicode strings, the backslash is used to escape Unicode characters using their hexadecimal representation.

```
unicode_char = '\u03B1'  # Represents the Greek letter alpha (α)
```

Raw Strings : By using the `r` prefix before a string, you can create a raw string in which the backslash is treated as a literal character and not as an escape character.

```
raw_string = r'C:\Users\John\Documents'
```

In summary, the backslash '\' is a versatile character used for various purposes in Python, primarily for escaping special characters and creating escape sequences in strings. Its meaning can vary depending on the context in which it is used**.**

7 Give an example of the following conditions:

a) Homogeneous list

b) Heterogeneous set

c) Homogeneous tuple

homogeneous list `list_1= [1, 2, 3, 4, 5]`
Heterogeneous set `set_1={1,'name',2.45}`
Homogeneous tuple `tuple_1=(1,2,5,8,9)`

8 Explain the mutable and immutable data types with proper explanation & examples.

data types can be categorized as either mutable or immutable based on whether the values of those types can be changed (mutated) after they are created. Understanding the distinction between mutable and immutable data types is essential because it affects how you work with and manipulate data in your Python programs. mutable data types allow you to modify their contents after creation, while immutable data types do not.

Mutable Data Types

List: Lists are mutable in Python. This means we can change, add, or remove elements in a list after it is created.

```
my_list = [1, 2, 3]
my_list[0] = 10             # Modify an element
my_list.append(4)           # Add an element
my_list.remove(2)           # Remove an element
```

Dictionary: Dictionaries are also mutable. we can modify the values associated with keys, add new key-value pairs, or remove existing key-value pairs.

```
my_dict = {'name': 'Alice', 'age': 30}
my_dict['age'] = 31                  # Modify a value
my_dict['city'] = 'New York'         # Add a new key-value pair
del my_dict['age']                   # Remove a key-value pair
```

Set: Sets are mutable as well. We can add and remove elements from a set.

```
my_set = {1, 2, 3}
my_set.add(4)    # Add an element
my_set.remove(2) # Remove an element
```

Immutable Data Types

Tuple: Tuples are immutable, which means we cannot change their contents once they are created. You can access elements, but you cannot modify or add new elements**.**

```
my_tuple = (1, 2, 3)
element = my_tuple[0]          # Access an element
 my_tuple[0] = 10              # Attempting to modify  would result in an error
```

String: Strings are immutable**.** we can access individual characters or substrings, but you cannot change them**.**

```
my_string = "Hello, World!"
char = my_string[0]          # Access a character
my_string[0] = 'J'           # Attempting to modify  would result in an error
```

Integer, Float, Boolean: Basic numeric types and Booleans are also immutable. You cannot change their values directly.

```
my_int = 42
my_int = 10

# Attempting to modify would create a new integer, not modify the original one
```

Frozen Set: A frozen set is an immutable version of a set. Once created, you cannot change its elements.

```
my_frozen_set = frozenset({1, 2, 3})
# Attempting to add or remove elements from my_frozen_set would result in an error
```

9 Write a code to create the given structure using only for loop.

```
    *
   ***
  *****
 *******
*********
```

```
n = 5                                          # Number of rows
for i in range(1, n + 1):                      # First loop for rows
   for j in range(n, i, -1):                   # Print spaces in decreasing order
      print(' ', end='')
   for k in range(1, 2 * i):                   # Print asterisks in increasing order
      print('*', end='')
   print()                                     # Move to the next line for the next row
```

10 Write a code to create the given structure using while loop.

```
|||||||||
 |||||||
  |||||
   |||
    |
```

```
n = 5
for i in range(n,0,-1):
    for j in range(0,n-i):
        print(" ",end="")
    for k in range(1,2*i):
        print("|",end="")
    print("")
```