Republic of the Philippines
BOHOL ISLAND STATE UNIVERSITY
Magsija, Balilihan, 6342, Bohol, Philippines
Office of Instruction
Balance | Integrity | Stewardship | Uprightness

Graphics and Visual Computing

WEEK 2: Python Basics Refresher

I. Learning Outcomes

By the end of this session, students should be able to:

1. Review of data types, control flow, functions, and object-oriented programming.

II. Introduction to Python

Python is a powerful and open-source programming language useful for developing software across multiple operating systems. It is object-oriented and provides a simple, easy-to-read, and user-friendly language, which enhances creativity. Additionally, it is beneficial to learn Python because it has numerous advantages.

Uses of Python:

1. To process the image.
2. To write Internet scripts.
3. To embed scripts.
4. To manipulate database programs.
5. To provide system utilities.
6. To create artificial intelligence (AI).
7. To create graphical user interface applications using IDEs on Windows and other platforms.

Advantages of Python:

1. Learn Python easily because the syntax and programming language are simple.
2. Prepare codes readily that can be used in various operating systems such as Linux, Windows, Unix, and Mac OS X.
3. Promptly access the Python standard library that helps users in creating, editing, accessing, running, and maintaining files.
4. Integrate programs and systems promptly because the programming language is easy to follow.
5. Handle the errors more reliably because the syntax can identify and raise exceptions.
6. Learn more quickly because the programming language is object-oriented.
7. Access IDLE, which makes it possible for users to create codes and check if the codes work, through Python's interactive systems.
8. Download Python for Free and enjoy all the benefits of a free application.
9. Embed your Python data in other systems.
10. Stop worrying about freeing the memory for your code, because Python does it automatically.

Graphics and Visual Computing

III. Basic Terms in Python

1. **Strings** – values enclosed inside double, single quotes, or triple quotes. They can be a word/text or a group of words, or a Unicode, or others. The advantage of the double quotes is that you can include values within the double quotes. The triple quote, on the other hand, signifies lengthy strings. They are useful to avoid getting an EOL (Endo f the Line) error.
2. **Variables** – containers for the string. In Python, they are usually objects. These can be numbers or strings. Remember that you have to declare the variables prior to using them.
3. **Statements** – stated sentences or syntax used to call a function to compute, to write a value, or other procedures needed in executing or performing Python commands.
4. **Lists** – collections of items that can include any variables. They are like arrays, with the variables usually enclosed in brackets, and the items or values separated by commas.
5. **Loops** – are statements that can be performed or executed one after the other – repeatedly, or once. There are two general types of loops, the 'for' and the 'while' loops.
6. **Function**
   - a code block that executes some functions or logic.

     ```
     print("Hello World!")    →    Output: Hello World!
     pow (8, 2)               →    Output: 64
     ```

   - a block of code that is defined outside of any class. It can be called by its name and can take parameters and return values. Functions are independent and do not belong to any object or class.

     ```
     def add(a, b):
         return a + b

     print(add(4, 5))         →    Output: 9
     ```

7. **Method** – is a function that is defined inside a class and is associated with an object. Methods can access and modify the data of the object they belong to. They always take at least one parameter, **self**, which refers to the instance of the class.

   ```
   class Calculator:
       def add(self, a, b):
           return a + b

   calc = Calculator()
   print(calc.add(2, 3))                    →    Output: 9
   ```

8. **Classes** – are groups of related data, like strings, integers, and lists that use related functions. To introduce or identify a class, use the function word 'class'.
9. **Objects** – are used extensively in Python because it is an object-oriented language. This allows users to name their files based on their group or individual value.
10. **Concatenation** – a series of connected strings or variables used in a Python program.

Graphics and Visual Computing

IV. Types of Python Statements

Several types of Python statements help create Python syntax/code:

1. **Simple statements** – composed of a single logical line. They may occur singly or in several statements in one line. If there are several simple statements, they can be separated by a semicolon. The most common simple statement is the 'print' statement.

```
print("hi")
```

2. **Assignment statements** – used when names are assigned to values, and when you want to modify mutable (can be changed) objects. The syntax is like that of expression statements.

```
myname = "apple"
```

3. **Expression statements** – generally used for computations and for evaluating an expression list. They are also useful in writing values. They usually return the (none) value, when used to call a procedure.

```
sum = 3 + 9
```

4. **import statements** – used to import files, functions or modules. Python has packages (directories) containing modules (files). You can quickly import modules by using the key 'import'.

```
Import name1
```

5. **continue statement** – indicates that a statement, usually a loop, continues with the next loop.

```
while (True):
      if (a = 1):
           continue
```

6. **break statement** – they '*break*' the nearest enclosing loop and resume execution on the next statement. But the loop will finally 'break' when the 'try' statement and the 'finally' clause are executed.

```
while (True):
      if (a != 1):
           break
```

7. **return statement** – usually used in evaluating an expression list and exiting a function, operation, or method. There are two forms: the '*return*' and '*return expressions*'. They could be present in the definition of a function, but not in the definition of a nested class.

```
return "Happy"
```

8. **if, elif, and else statements** – conditional statements that require checkpoints to be satisfied for their following statements to be satisfied.

V. Basic Elements of Python

To facilitate a better understanding of Python, here are its basic elements. Remember, there may be a slight variation with the different Python versions.

1. **<u>Variables</u>**

Python uses information that is not constant. With that, think of variables as a temporary container to keep the data. Creating a variable in Python is simple. You assign

Graphics and Visual Computing

a value to a variable name and start using it. When using these, be descriptive of the naming conventions. Additionally, Python figures out the variable type automatically.

Variable Naming Rules

- Variable names can contain letters (both uppercase and lowercase), digits, and underscores.
- They must begin with a letter or an underscore (_), but not with a digit.
- They are case sensitive (myVar and myvar are different variables)
- Python keywords cannot be used as variable names.

Example

```
# Make the naming conventions of the variables as descriptive
# assigning variable names
myVar = 10
my_var = 6
_var = 6
myvar = 6
```

2. **Data Types**
   - ***Numeric Data Types***
     1. **int**: Whole numbers, either positive or negative (e.g., 100, 5, -3, 1000).
     2. **float**: Numbers representing decimal numbers (e.g., 3.14, -0.1, 2.0).
     3. **complex**: Complex numbers (e.g., 3 + 4j).

Example

```
# Example for int
num1 = 4
num2 = 5
print("Num1 and Num2: ", num1, num2)
print("Typeof Num1: ", type(num1))
print("Typeof Num2: ", type(num2))

# Example for float
num3 = 8.5
num4 = -12.4
print("Num1 and Num2: ", num3, num4)
print("Typeof Num3: ", type(num3))
print("Typeof num4: ", type(num4))
```

Graphics and Visual Computing

```
15  # Example of complex
16  num5 = 2.9
17  num6 = 1 + 2*num5
18  print("Num6: ", num6)
19  print("Typeof num6: ", type(num6))
✓ 0.0s
```

```
Num1 and Num2:  4 5
Typeof Num1:  <class 'int'>
Typeof Num2:  <class 'int'>
Num1 and Num2:  8.5 -12.4
Typeof Num3:  <class 'float'>
Typeof num4:  <class 'float'>
Num6:  6.8
Typeof num6:  <class 'float'>
```

- ***Sequence Data Types***
  1. **str**: This are phrases or words enclosed in "" or '', representing texts or characters ("Hello World!", 'apple', etc).
  2. **list**: Used to create a collection, able to be modified later on. Can be of same or different types ([1, 2, 3, 4], ["Apple", "Banana", "Camote"], etc). We will be using list indexing to access a specific value. We will be starting with index 0.
  3. **tuple**: Fixed collection of elements or immutable sequence of elements (unchangeable); enclosed in () (e.g., ("Emman", "Nanay", "Tatay"), etc).

Example

```
1  # Example of str
2  mydog = "Sunny"
3  mycat = 'Mingming'
4  print("Dog: ", mydog)
5  print("Cat: ", mycat)
6
7  # Example of list
8  myhobbies = ["reading", "eating", "sleeping"]
9  favnumbers = [1, 8, 11, 18]
10 veryrandom = ["A", 'mixed of', 1, 3.2, -10, True]
11 print("My Hobbies: ", list(myhobbies))
```

Graphics and Visual Computing

```
12  print("My Fav Numbers: ", favnumbers)
13  print("Very Random: ",veryrandom)
```
✓ 0.0s

```
Dog:  Sunny
Cat:  Mingming
My Hobbies:  ['reading', 'eating', 'sleeping']
My Fav Numbers:  [1, 8, 11, 18]
Very Random:  ['A', 'mixed of', 1, 3.2, -10, True]
```

- ***Mapping Data Type***
  1. **dict**: Dictionaries enclosed in {}, are key-value pair; has a key and the associated value in it (e.g., {"key": "value"}).

Example

```
1  # Example of dictionary
2  person = {"name": "Alice",
3            "age": 25}
4  print(f"person is a dictionary: {person}")
```
✓ 0.0s

```
person is a dictionary: {'name': 'Alice', 'age': 25}
```

- ***Boolean Type***
  1. **bool**: Boolean values that return true or false

Example

```
1  # Example of boolean
2  issunny = True
3  israiny = False
4
5  print("Is it sunny? ", issunny)
6  print("Is it raining? ", israiny)
```
✓ 0.0s

```
Is it sunny?  True
Is it raining?  False
```

Graphics and Visual Computing

3. **Output and Input**

Every computer program needs a way to communicate with its user. The user provides input through encoding, and the program processes this input to produce the output, which is then presented to the user.

Python offers functions to accept input from users via the **input()** function. This function reads a line of text entered by the user and always returns it as a string. To use a specific data type, you need to convert the input accordingly.

Example

```
1  name = input("Enter your name: ")
2  age = int(input("Enter your age: ")) # Convert to integer
```

Python also provides functions and techniques for displaying output using the **print()** function. This function prints objects or text to the standard output (console) and can accept multiple arguments, as well as options for specifying the separator (sep) and the ending (end).

Example

```
1  print("Hello, World!") # Simple text output
2  print("Name:", "Alice", "Age:", 25, sep=" | ") # Custom
   separator
3  print("Goodbye", end="!") # Custom end character
```

VI. Python Operations and Expressions

In Python programming, operators are crucial for performing various operations on values and variables. They are the fundamental elements for creating robust and functional code.

1. **Operators**

Operators are special symbols used to carry out logical and arithmetic operations. Below are some common types of operators with examples:

1. **Arithmetic Operators**
   - Addition (+): Adds two operands.
   - Subtraction (-): Subtracts the second operand from the first.
   - Multiplication (*): Multiplies two operands.
   - Division (/): Divides the first operand by the second.
   - Example: a + b, a - b, a * b, a / b
2. **Comparison Operators**
   - Equal to (==): Checks if two operands are equal.
   - Not equal to (!=): Checks if two operands are not equal.
   - Greater than (>): Checks if the first operand is greater than the second.
   - Less than (<): Checks if the first operand is less than the second.
   - Example: a == b, a != b, a > b, a < b

Graphics and Visual Computing

3. **Logical Operators**
   - AND (and): Returns True if both statements are true.
   - OR (or): Returns True if one of the statements is true.
   - NOT (not): Reverses the result, returns False if the result is true.
   - Example: a and b, a or b, not a
4. **Assignment Operators**
   - Assign (=): Assigns a value to a variable.
   - Add and Assign (+=): Adds and assigns the result.
   - Subtract and Assign (-=): Subtracts and assigns the result.
   - Example: a = b, a += b, a -= b

2. **Operands**

Operands are the values or variables upon which operators perform their operations. For instance, in the expression a + b, a and b are operands, and + is the operator.

Example

```
# Using arithmetic operators
a = 10
b = 5
print(a + b)  # Output: 15
print(a - b)  # Output: 5
print(a * b)  # Output: 50
print(a / b)  # Output: 2.0

# Using comparison operators
print(a == b)  # Output: False
print(a != b)  # Output: True

# Using logical operators
print(a > b and a < 20)  # Output: True
print(a > b or a < 5)    # Output: True
print(not(a > b))        # Output: False

```

Graphics and Visual Computing

```
18  # Using assignment operators
19  a = 10
20  a += 5
21  print(a)  # Output: 15
22  a -= 3
23  print(a)  # Output: 12
24
```

3. **Expressions**

An expression is a combination of operators and operands that is interpreted to produce a value. In Python, as in any programming language, the precedence of operators determines the order in which the operations within an expression are performed. Let's explore the different types of expressions in Python with examples.

Types of Expressions

1. **Constant Expressions** – the expressions that have constant values only.
   - x **=** 15 **+** 1.3
2. **Arithmetic Expressions** – involve arithmetic operators and produce numerical results.
   - a + b - c * d / e
3. **Relational Expressions** – compare values and produce Boolean results (True or False).
   - a < b, a == b
4. **Logical Expressions** – combine multiple conditions and produce Boolean results.
   - a and b, a or b, not a
5. **Bitwise Expressions** – computations are performed at the bit level.
   - a **=** a >> 2
6. **Compound Expressions** – a combination of multiple types of expressions.
   - Example: (a + b) * c, a < b and c > d

VII. Built-in Functions

Python offers a rich set of built-in functions that are ready to use without needing any import or setup. These functions simplify common tasks such as type conversion, mathematical operations, string manipulation, and more.

1. **Type Conversion Functions**

These functions convert data from one type to another.

- **int()** – converts a value to an integer.
- **float()** – converts a value to a floating-point number.
- **str()** – converts a value to a string.

Graphics and Visual Computing

- **bool()** – converts a value to a Boolean.

2. **Mathematical Functions**
   These functions perform various mathematical operations.
   - **abs()** – returns the absolute value of a number.
   - **round()** – rounds a number to the nearest integer.
   - **pow()** – returns the value of a number raised to the power of another.
3. **String Functions**
   These functions perform operations on strings.
   - **len()** – returns the length of a string.
   - **upper()** – converts a string to uppercase.
   - **lower()** – converts a string to lowercase.
4. **Utility Functions**
   These functions perform miscellaneous operations.
   - **type()** – returns the type of an object.
   - **print()** – prints a value to the console.
   - **input()** – reads a value from the console.
5. **Sequence Functions**
   These functions perform operations on sequences like lists, tuples, etc.
   - **min()** – returns the smallest item in an iterable.
   - **max()** – returns the largest item in an iterable.
   - **sum()** – returns the sum of all items in an iterable.

VIII. Commonly Used Methods in Python

Methods in Python are functions that are associated with objects. They provide a way to manipulate and interact with data stored in objects, such as strings, lists, and dictionaries. Knowing how to use these methods effectively can greatly enhance your ability to write clean and efficient code.

1. **String Methods**
   1. **upper()** – converts all characters in a string to uppercase.
      ```
      text = "hello"
      print(text.upper())          →     Output: HELLO
      ```
   2. **lower()** – converts all characters in a string to lowercase.
      ```
      text = "HELLO"
      print(text.lower())          →     Output: hello
      ```
   3. **strip()** – removes leading and trailing whitespace from a string.
      ```
      text = " hello "
      print(text.strip())          →     Output: hello
      ```
   4. **replace()** – replaces a substring with another substring.
      ```
      text = "hello world"
      print(text.replace("world", "Python")
                  →     Output: hello Python
      ```

Graphics and Visual Computing

5. **split()** – splits a string into a list using a specified delimiter.

```
text = "hello world"
print(text.split())
                → Output: ["hello", "world"]
```

2. **List Methods**
   1. **append()** – adds an element to the end of the list.

```
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)          → Output: [1, 2, 3, 4]
```

   2. **remove()** – removes the first occurrence of a specified element.

```
numbers = [1, 2, 3, 2]
numbers.remove(2)
print(numbers)          → Output: [1, 3, 2]
```

   3. **pop()** – removes and returns the element at the specified index.

```
numbers = [1, 2, 3]
numbers.pop(1)
print(numbers)          → Output: [1, 3]
```

   4. **sort()** – sorts the list in ascending order.

```
numbers = [3, 1, 2]
numbers.sort()
print(numbers)          → Output: [1, 2, 3]
```

   5. **reverse()** – reverses the order of the list.

```
numbers = [1, 2, 3]
numbers.reverse()
print(numbers)          → Output: [3, 2, 1]
```

3. **Dictionary Methods**
   1. **keys()** – returns a view object that displays a list of all the keys.

```
my_dict = {"name": "Alice", "age": 30}
print(my_dict.keys())
                → Output: dict_keys(["name", "age"])
```

   2. **values()** – returns a view object that displays a list of all the values.

```
my_dict = {"name": "Alice", "age": 30}
print(my_dict.values())
                → Output: dict_values(["Alice", 30])
```

   3. **items()** – returns a view object that displays a list of dictionary's key-value tuple pairs.

```
my_dict = {"name": "Alice", "age": 30}
print(my_dict.items())
→ Output: dict_items([("name", "Alice"), ("age", 30)])
```

Graphics and Visual Computing

4. **update()** – updates the dictionary with elements from another dictionary or an iterable of key-value pairs.

```
my_dict = {"name": "Alice", "age": 30}
my_dict.update({"location": "Wonderland"})
print(my_dict)
```

→ `Output: {"name": "Alice", "age": 30, "location": "Wonderland"}`

5. **get()** – returns the value for the specified key if key is in dictionary.

```
my_dict = {"name": "Alice", "age": 30}
print(my_dict.get("name"))
                → Output: "Alice"
print(my_dict.get("location", "Not Found"))
                → Output: "Not Found"
```

IX. Assessment Tasks

- **Quiz**
- **Activity**
- **Assignment**