Department of Information Technology
Statistics with Python

Module 1: Introduction to Python - Installing Python, The concept of data types; variables, assignments; arithmetic operators and expressions; Input and Output in Python

Data Types in Python

According to the properties they possess, there are mainly six python data types. Although there is one more data type range which is often used while working with loops in python.

a) Python Numeric Data type

In Python, numeric data type is used to hold numeric values.
Integers, floating-point numbers and complex numbers fall under Python numbers category.
They are defined as `int`, `float` and `complex` classes in Python.

`int` - holds signed integers of non-limited length.
`float` - holds floating decimal points and it's accurate up to 15 decimal places.
`complex` - holds complex numbers.
We can use the `type()` function to know which class a variable or a value belongs to.

Let's see an example,

```
num1 = 5
print(num1, 'is of type', type(num1))

num2 = 2.0
print(num2, 'is of type', type(num2))

num3 = 1+2j
print(num3, 'is of type', type(num3))
```

Run Code

Output

```
5 is of type <class 'int'>
2.0 is of type <class 'float'>
```

```
(1+2j) is of type <class 'complex'>
```

In the above example, we have created three variables named num1, num2 and num3 with values **5**, **5.0**, and 1+2j respectively.

We have also used the type() function to know which class a certain variable belongs to. Since,
5 is an integer value, type() returns int as the class of num1 i.e <class 'int'>
2.0 is a floating value, type() returns float as the class of num2 i.e <class 'float'>
1 + 2j is a complex number, type() returns complex as the class of num3 i.e <class 'complex'>

b) Python List Data Type

List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ]. For example,

```
languages = ["Swift", "Java", "Python"]
```

Here, we have created a list named languages with **3** string values inside it.
To access items from a list, we use the index number **(0, 1, 2 ...)**. For example,

```
languages = ["Swift", "Java", "Python"]

# access element at index 0
print(languages[0])  # Swift

# access element at index 2
print(languages[2])  # Python
```

In the above example, we have used the index values to access items from the languages list.
languages[0] - access first item from languages i.e. Swift
languages[2] - access third item from languages i.e. Python

c) Python Tuple Data Type

Tuple is an ordered sequences of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

In Python, we use the parentheses () to store items of a tuple. For example,

```
product = ('Xbox', 499.99)
```

Here, product is a tuple with a string value Xbox and integer value **499.99**.

Similar to lists, we use the index number to access tuple items in Python . For example,

```
# create a tuple
product = ('Microsoft', 'Xbox', 499.99)

# access element at index 0
print(product[0])  # Microsoft
```

```
# access element at index 1
print(product[1])   # Xbox
```

d) Python String Data Type

String is a sequence of characters represented by either single or double quotes. For example,

```
name = 'Python'
print(name)

message = 'Python for beginners'
print(message)
```

Output

```
Python
Python for beginners
```

In the above example, we have created string-type variables: `name` and `message` with values `'Python'` and `'Python for beginners'` respectively.

e) Python Set Data Type

Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces `{ }`. For example,

```
# create a set named student_id
student_id = {112, 114, 116, 118, 115}

# display student_id elements
print(student_id)

# display type of student_id
print(type(student_id))
```

Output

```
{112, 114, 115, 116, 118}
<class 'set'>
```

Here, we have created a set named `student_info` with **5** integer values.
Since sets are unordered collections, indexing has no meaning. Hence, the slicing operator `[ ]` does not work.

f) Python Dictionary Data Type

Python dictionary is an ordered collection of items. It stores elements in key/value pairs. Here, keys are unique identifiers that are associated with each value.

Let's see an example,

```
# create a dictionary named capital_city
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
print(capital_city)
```

Output

```
{'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
```

In the above example, we have created a dictionary named `capital_city`. Here,
Keys are `'Nepal'`, `'Italy'`, `'England'`
Values are `'Kathmandu'`, `'Rome'`, `'London'`

Access Dictionary Values Using Keys -We use `keys` to retrieve the respective `value`. But not the other way around. For example,

```
# create a dictionary named capital_city
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
print(capital_city['Nepal'])  # prints Kathmandu
print(capital_city['Kathmandu'])  # throws error message
```

Here, we have accessed values using keys from the `capital_city` dictionary.
Since `'Nepal'` is key, `capital_city['Nepal']` accesses its respective value i.e. `Kathmandu`

However, `'Kathmandu'` is the value for the `'Nepal'` key, so `capital_city['Kathmandu']` throws an error message.

Creating Python Variables

Python variables do not need explicit declaration to reserve memory space or you can say to create a variable. A Python variable is created automatically when you assign a value to it. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example −

```
counter = 100          # Creates an integer variable
miles   = 1000.0       # Creates a floating point variable
name    = "Zara Ali"   # Creates a string variable
```

Once we create a Python variable and assign a value to it, we can print it using **print()** function. Following is the extension of previous example and shows how to print different variables in Python:

```
print (counter)
print (miles)
print (name)
```

Here, 100, 1000.0 and "Zara Ali" are the values assigned to *counter*, *miles*, and *name* variables, respectively. When running the above Python program, this produces the following result −

```
100
1000.0
Zara Ali
```

Delete a Variable

You can delete the reference to a number object by using the del statement. The syntax of the del statement is −

```
del var1[,var2[,var3[....,varN]]]
```

You can delete a single object or multiple objects by using the del statement. For example −

Example

Following examples shows how we can delete a variable and if we try to use a deleted variable then Python interpreter will throw an error:

```
counter = 100
print (counter)
del counter
print (counter)
```

This will produce the following result:

```
100
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    print (counter)
NameError: name 'counter' is not defined
```

Assignment

Python allows you to assign a single value to several variables simultaneously which means you can create multiple variables at a time. For example −

```
a = b = c = 100
print (a)
print (b)
print (c)
```

This produces the following result:

```
100
100
100
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example

```
a,b,c = 1,2,"Zara Ali"
print (a)
print (b)
print (c)
```

This produces the following result:

```
1
2
Zara Ali
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "Zara Ali" is assigned to the variable c

---

Arithmetic operators and expressions in Python

Operators are used to perform operations on variables and values.
In the example below, we use the + operator to add together two values:

Example

```
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

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

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

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

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)

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

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	

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

---

Input and Output in Python

Sometimes a developer might want to take user input at some point in the program. To do this Python provides an input() function.

Syntax:
input('prompt')

Example: Python get user input with a message

```
# Taking input from the user
name = input("Enter your name: ")
# Output
print("Hello, " + name)
```

Output:
Enter your name: GFG
Hello, GFG

Example 1: Python Print Statement

```
print('Good Morning!')
print('It is rainy today')
```

Run Code

Output

```
Good Morning!
It is rainy today
```

Example 2: Python print() with sep parameter

```
print('New Year', 2023, 'See you soon!', sep= '* ')
```

Output

```
New Year* 2023* See you soon!
```

Sources:
https://www.w3schools.com/python/python_operators.asp

https://www.programiz.com/python-programming/variables-datatypes

https://www.programiz.com/python-programming/input-output-import