Variables and Data Types

Variables

A variable is a name given to any storage area or memory location in a program.

In simple words, we can say that a variable is a container that contains some information, and whenever we need that information, we use the name of that container to access it.

Variables are a major part of computer programming. Variables allow you to store a value that you can use later in the program. You can therefore make the program do different things, depending on the value of the variable.

Creating a variable

```
a = "Hello World"
b = 23
c = 56.66
print(a)
print(b)
print(c)
print(b + c)
```

Output

```
Hello World
23
56.66
79.66
```

In this example we have created 3 variables and named them **a**, **b** and **c.**

We have assigned **“Hello World”** to **a**, **23** to **b** and **56.66** to **c.**

In Python you do not need to declare the variable before assigning a value to it. You also do not need to declare its data type. This contrasts with some other languages (such as Java, C, C++, etc.), where you must declare the variable first and explicitly state its data type.

After declaring, you can also output or print them as usual using the **print** statement.

Multiple assignment (Assigning multiple variables in single line)

In Python, you can assign multiple variables within a single statement.

The above code can be changed as shown below and will work perfectly fine.

```
a, b, c = "Hello World", 23, 56.66
print(a)
print(b)
print(c)
print(b + c)
```

Output

```
Hello World
23
56.66
79.66
```

Rules for naming a variable

A variable name is an **identifier**. In Python, an identifier can be a combination of letters in lowercase (a to z), uppercase (A to Z), digits (0 to 9), and the underscore (_) character.

Some valid examples:

Here are some variable names that are perfectly fine and won't give us any errors.

```
name = "Cristiano"
Name = "Cristiano"
Name1 = "Cristiano"
Name_1 = "Cristiano"
_name = "Cristiano"
_student_first_name = "Cristiano"
```

Cannot start with number

You cannot start the variable name with a number, it will give us an error.

```
# We cant name variable like this
1name="Cristiano"
```

Cannot use a Python reserved Keyword

You cannot use any Python keywords as your identifier name. Here are the Python keywords:

False	def	if	raise	None
del	import	return	True	elif
in	try	and	else	is
while	as	except	lambda	with
assert	finally	nonlocal	yield	break
for	not	class	from	or
continue	global	pass		

Case sensitive

In Python, variable names (as with all identifiers) are case sensitive. Therefore, the following are two different variables:

```
firstname = "Cristiano"
Firstname = "Melon Eusk"
print(firstname)
print(Firstname)
```

Outputting will give us two different values because they are 2 different variables.

```
Cristiano
Melon Eusk
```

Data types in Python

Data types are the classification or categorization of knowledge items. It represents the type useful that tells what operations are often performed on specific data. Since everything is an object in Python programming, data types are classes and variables are instances (object) of those classes.

Data types are an important concept within the python programing language. Every value has its own python data type, in the python programming language. The classification of knowledge items or to place the info value into data category is named Data Types. It helps to know what quiet operations are often performed on a worth.

A data type, in programming, is a classification that specifies which type of value a variable has. Primarily there are following data types in Python.

- Integers (<class **'int'**>): Used to store integers
- Floating point numbers (<class **'float'**>): Used to store decimal or floating-point numbers
- Strings (<class **'str'**>): Used to store strings
- Booleans (<class **'bool'**>): Used to store True/False type values
- **None**: None is a literal to describe 'Nothing' in Python
- List
- Tuple
- Set
- Dictionary

Example

```
# Variable in Python:
abc = "It's a string variable"
_abcnum = 40  # It is an example of int variable
abc123 = 55.854  # It is an example of float variable
print(_abcnum + abc123)  # This will give sum of 40 + 55.854
```

In the above code you can see that there are 3 different variables created with 3 different values.

- **abc** is storing a **String** data type.
- **_abcnum** is storing an **Integer** data type.
- **abc123** is storing a **Float** data type.

Determining a Variable's Type

Whenever we create a variable and assign value to it, we have an option to print the datatype stored into that variable.

You can use the type() function to find out what type a variable is. You can also pass the type() function to a print() function to print the result to the screen.

Example

```
# Set the variables
firstName = "John Doe"
age = 44

# Display their type
print(type(firstName))
print(type(age))
```

Output

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

This tells us that **firstName** variable has a **string** stored in it and **age** variable has an **int** stored in it.

Note – We can't do arithmetic operations of numbers with strings i.e. we can't add a string to any number. Have a look at the example below:

```
var1 = "It's a String"
var2 = 5
print(var1 + var2)   # It will give an
# error as we can't add string to any number.
```

Output

```
Traceback (most recent call last):
  File "C:\Users\aniru\Desktop\Python Programming\hello.py", line 3, in <module>
    print(var1 + var2)  # It will give an
TypeError: can only concatenate str (not "int") to str
```

Note – We can add (concatenate) two or more strings and the strings will be concatenated to return another string. Here's the example showing that:

```
var1 = "My Name is "
var2 = "xyz"
var3 = var1 + var2 + " & I am a Good Boy."
print(var1 + var2)
print(var3)
```

Output

```
My Name is xyz
My Name is xyz & I am a Good Boy.
```

Typecasting

Typecasting is the way to change one data type of any data or variable to another datatype, i.e., it changes the data type of any variable to some other data type.

We know it is a bit confusing but let me tell you in a simple manner. Suppose there is a string "34" Note: String is not integer since it is enclosed in double-quotes) and as we know we cannot add this to an integer number let us say 6. But to do so we can typecast this string to int data type and then we can add 34+6 to get the output as 40. Have a look at the program below:

Example

```
# Typecasting in Python:
abc = 5
abc2 = '45'
abc3 = 55.95
xyz = 5.0
abc4 = int(abc2)
print(abc + abc4)
print(abc + int(abc2))
print(float(abc) + xyz)
```

Output

```
50
50
10.0
```

There are many functions to convert one data type into another type:

- **str()** – this function allows us to convert some other data type into a string.
- **int()** – this function allows us to convert some other data type into an integer. For example, **int("34")** returns 34 which is of type integer (int)
- **float()** – this function allows us to convert some other data type into floating-point number i.e. a number with decimals.

Input Function

This function allows the user to receive input from the keyboard into the program as a string.

input() function always takes input as a string i.e. if we ask the user to take a number as input even then it will take it as a string and we will have to typecast it into another data type as per the use case.

If you enter 45 when input() is called, you will get "45" as a string.

Example

```
# Input Function in Python:
print("Enter your name : ")
name = input()  # It will take input from user
print("Your Name is", name)  # It will show the name
xyz = input("Enter your age : ")
print("Your age is", xyz)
```

Output

```
Enter your name :
Airudh Khurana
Your Name is Airudh Khurana
Enter your age : 44
Your age is 44
```

Example

Ask 2 numbers from user and print the sum of those numbers.

Let us solve this question in different style showing you common mistakes.

Ans 1 (Does not give expected answer)

```
a = input("Enter number 1 = ")
b = input("Enter number 2 = ")
print(a + b)
```

```
Enter number 1 = 33
Enter number 2 = 46
3346
```

This solution does not work because, whenever we take input from user, it's data type will always be a **string**. And we know when we add 2 **strings**, the result will always be a concatenation of 2 strings.

Ans 2

```
a = input("Enter number 1 = ")
b = input("Enter number 2 = ")
c = int(a)
d = int(b)
print(c + d)
```

```
Enter number 1 = 33
Enter number 2 = 46
79
```

This will give us proper solution because we converted our both **strings** into **integers** and stored them in **c** and **d**.

Ans 3

```
a = int(input("Enter number 1 = "))
b = int(input("Enter number 2 = "))
print(a + b)
```

This solution will also work because we directly converting the string at the same line as input.

String formatting in Python

String formatting is the process of infusing things in the string dynamically and presenting the string.

There are different ways to perform string formatting in Python:

- Formatting with % Operator.
- Formatting with string literals, called f-strings.

Formatting with % Operator

It is the oldest method of string formatting. Here we use the modulo % operator. The modulo % is also known as the "string-formatting operator."

Let us print the **name** and **age** of user in a proper way using **%**.

Example

```
name = input("Enter your name = ")
age = int(input("Enter your age = "))
print("Your name is %s" % name)
print("Your age is %d" % age)
print("Your name is %s and your age is %d" % (name, age))
```

Output

```
Enter your name = Anirudh
Enter your age = 33
Your name is Anirudh
Your age is 33
Your name is Anirudh and your age is 33
```

'%s' is used to inject strings similarly '%d' for integers, '%f' for floating-point values, '%b' for binary format.

Try these examples below:

```
pi = 3.14563
print("The value of pi is: %f" % pi)
print("The value of pi up to 2 decimal is: %.2f" % pi)
```

Formatted String using F-strings

PEP 498 introduced a new string formatting mechanism known as Literal String Interpolation or more commonly as F-strings (because of the leading f character preceding the string literal). The idea behind f-strings is to make string interpolation simpler.

To create an f-string, prefix the string with the letter " f ". The string itself can be formatted in much the same way that you would with str.format(). F-strings provide a concise and convenient way to embed python expressions inside string literals for formatting.

Example

```
name = input("Enter your name = ")
age = int(input("Enter your age = "))
print(f"Your name is {name}")
print(f"Your age is {age}")
print(f"Your name is {name} and your age is {age}")

pi = 3.14563
print(f"The value of pi is: {pi}")
```

Output

```
Enter your name = Anirudh
Enter your age = 33
Your name is Anirudh
Your age is 33
Your name is Anirudh and your age is 33
The value of pi is: 3.14563
```

Practice Examples (BASICS)

1. Write a program that takes two numbers as input from the user and then prints the sum of these numbers.
2. Write a Program that takes Length and Breadth as input from user and print the Area of Rectangle.
3. Ask 3 numbers from User and Calculate the Average.
4. Calculate sum of 5 subjects and Find percentage.
5. Ask marks in 5 subjects and calculate Total Marks and Find percentage.
6. Ask value in Rupees and Convert into Paise.
7. Calculate Area of Square by taking Length from User.
8. Ask number of games played in a tournament. Ask the user number of games won and number of games loss. Calculate number of tie and total Points. (1 win= 4 points, 1 tie =2 points)

For solutions, check the end of the book.