AN7914 Week 01 Python

January 29, 2024

1 Week 1 Python

1.1 Simple Python Program

First we write a simple Python program thant prints `Hello World`.

[61]:
```
print("Hello World")
```

```
Hello World
```

You will notice we used a Python built-in function `print()`. It can take any valid argument/input. Our argument was `"Hello World"`. We could have done the same thing with a little variation.

[62]:
```
print('Hello World')
```

```
Hello World
```

Notice here that we got the same results. It printed `Hello World`. But do you see the difference?

We used single quotes `' '` instead of double quotes `" "`.

We can also print numbers.

[63]:
```
print(2)
```

```
2
```

[64]:
```
print(100)
```

```
100
```

1.2 Values and Types

A value is one of the basic things a program works with, like a letter or a number. Notice that we printed `Hello World`, `2` and `100`. These values belong to different types: `2` is an *integer*, and `"Hello, World!"` is a *string*, so called because it contains a "string" of letters. You (and Python) can identify strings because they are enclosed in quotation marks. Just use `type()` function and Python will tell the *type*.

[65]:
```
type('Hello World')
```

[65]:
```
str
```

[66]:
```
type(2)
```

[66]:
```
int
```

[67]:
```
type(2.0)
```

[67]:
```
float
```

[68]:
```
type(2.2)
```

[68]:
```
float
```

[69]:
```
type('2')
```

[69]:
```
str
```

1.3 Variables

One of the most powerful features of a programming language is the ability to manipulate *variables*. A variable is a name that refers to a value. An *assignment statement* creates new variables and gives them values:

[70]:
```
message = 'And now for something completely different'
some_number=2
```

The example above makes 2 assignments. The first assigns a string to a new variable named `message`; the second assigns the integer `2` to `some_number`. To display the value of a variable, you can use a print statement:

[71]:
```
print(message)
```

```
And now for something completely different
```

[72]:
```
print(some_number)
```

```
2
```

The *type* of a variable is the type of the value it refers to.

[73]:
```
type(message)
```

[73]:
```
str
```

[74]:
```
type(some_number)
```

[74]:
```
int
```

1.4 Print Function and Variables

In `print()` function we can introduce variables. Let's say we create variable called `name`.

[75]:
```
name="Nasreen"
```

[76]:
```
print("Hello",name)
```

```
Hello Nasreen
```

In the code above we used two inputs to the print function `"Hello"` and `name`. The second input is the `name` variable we created where we stored the value `Nasreen`. We could have gotten the same result by *concatanating*. See the following.

[77]:
```
print("Hello"+ name)
```

```
HelloNasreen
```

But the output is not desirable as there no space (or *white-space*) between Hello and Nasreen. We can fix that by putting a white space in `"Hello"` by typing `"Hello "`.

[78]:
```
print("Hello "+ name)
```

```
Hello Nasreen
```

1.4.1 Input Function

Now let's get some user input and then print Hello to the user.

[79]:
```
name_input=input("What is your name? ")
print("Hello",name_input)
```

```
What is your name? Sakib
Hello Sakib
```

We successfully managed to say hello the user Jimmy. Here first created a variable `name_input` where we stored the name of the user. We used a built-in Python function `input()` to get the user name. This built-in gets input from the keyboard. When this function is called, the program stops and waits for the user to type something. When the user presses **Return** or **Enter**, the program resumes and input returns what the user typed as a *string*.

1.4.2 Formatting Strings

Probably the most elegant way to use strings would be as follows:

[80]:
```
name_input=input("What is your name? ")
print(f"hello {name_input}")
```

```
What is your name? Sakib
hello Sakib
```

Notice the `f` in `print(f"hello, {name_input}")`. This `f` is a special indicator to Python to treat this string a special way, different than previous approaches we have illustrated in this lecture. Expect that you will be using this style of strings quite frequently in this course. In the example above we used curly brackets `{}`. When the print function is called python python is going to look for the variable inside the curly braket and print the value of that variable.

1.4.3 Removing unnecessary white-space

Look at the following program and its output.

[81]:
```
name_input=input("What is your name? ")
print("hello",name_input)
```

```
What is your name?     Sakib
hello     Sakib
```

We see that the user pressed the spacebar for too long!! Thus messing up our output. You should never expect your user will cooperate as intended. Therefore, you will need to ensure that the input of your user is corrected or checked. It turns out that built into strings is the ability to remove whitespace from a string.

By utilizing the method strip on `name_input` as `name_input_remove = name_input.strip()`, it will strip all the whitespaces on the left and right of the users input. You can modify your code to be:

[82]:
```
name_input=input("What is your name? ")
name_input_removespace=name_input.strip()
print("hello",name_input_removespace)
```

```
What is your name?    sakib
hello sakib
```

1.4.4 Title

Using the title method, it would title case the user's name:

[83]:
```
# Ask the user for their name
name = input("What's your name? ")

# Remove whitespace from the str
name = name.strip()

# Capitalize the first letter of each word
name = name.title()

# Print the output
print(f"Hello, {name}")
```

```
What's your name?    sakib
Hello, Sakib
```

Notice that you can modify your code to be more efficient:

[84]:
```
# Ask the user for their name
name = input("What's your name? ")

# Remove whitespace from the str and capitalize the first letter of each word
name = name.strip().title()

# Print the output
print(f"Hello, {name}")
```

```
What's your name?    sakib
Hello, Sakib
```

We could even go further!

[85]:
```
# Ask the user for their name, remove whitespace from the str and capitalize␣
  ↪the first letter of each word
name = input("What's your name? ").strip().title()

# Print the output
print(f"hello, {name}")
```

```
What's your name?    sakib
hello, Sakib
```

1.4.5 Comments

Comments are a way for programmers to track what they are doing in their programs and even inform others about their intentions for a block of code. In short, they are notes for yourself and others that will see your code! We have been using them in the previous couple of cells by putting a hashtag sign `#`. When python sees this hashtags it doesn't execute that line. You can add comments to your program to be able to see what it is that your program is doing (see the examples above).

1.5 Integers or `int`

In Python, an integer is referred to as an `int`. In the world of mathematics, we are familiar with +, -, *, /, and % operators. That last operator `%` or modulo operator may not be very familiar to you. You don't have to use the text editor window in your compiler to run Python code. Down in your terminal, you can run python alone. You will be presented with »> in the terminal window. You can then run live, interactive code. You could type 1+1 and it will run that calculation. This mode will not commonly be used during this course. But let's do some calculations here.

[86]:
```
2+3
```

[86]: 5

[87]:
```
a=2
b=3
```

```
print(a+b)
```

```
5
```

[88]:
```
print(a/b)
```

```
0.6666666666666666
```

[89]:
```
print(a*b)
```

```
6
```

Now if you want to use exponents like calculating 2^3 or $2 \times 2 \times \times 2$. You need to use `2**3`.

[90]:
```
2**3
```

[90]: 8

[91]:
```
print(a**b)
```

```
8
```

1.5.1 Converting to `int`

If you want to convert a string to integer we just use the `int()` function.

[92]:
```
c='2'
d='3'
c1=int(c)
d1=int(d)
print(c1*d1)
```

```
6
```

The following converts `float` to `int`.

[93]:
```
z=3.2
z1=int(3.2)
print(z1)
```

```
3
```

We have doing some calculations with python. But we can make it more interactive by using `input()`

[94]:
```
x = input("What's x? ")
y = input("What's y? ")

z = x + y
```

```
print(z)
```

```
What's x? 1
What's y? 2
12
```

Running this program, we discover that the output is incorrect as `12`. Why might this be?

Prior, we have seen how the `+` sign concatenates two strings. Because your input from your keyboard on your computer comes into the compiler as text, it is treated a string. We, therefore, need to convert this input from a string to an integer. We can do so as follows:

[95]:
```
x = input("What's x? ")
y = input("What's y? ")

z = int(x) + int(y)

print(z)
```

```
What's x? 1
What's y? 2
3
```

We can further improve our program as follows:

[96]:
```
x = int(input("What's x? "))
y = int(input("What's y? "))

print(x + y)
```

```
What's x? 1
What's y? 2
3
```

1.6 Creating Your Own Function

So far, we have only been using the functions that come with Python, but it is also possible to add new functions. A function definition specifies the name of a new function and the sequence of statements that execute when the function is called. Once we define a function, we can reuse the function over and over throughout our program. We can use the python's key `def` word to create our own function.

[97]:
```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
```

`def` is a keyword that indicates that this is a function definition. The name of the function is print_lyrics. The rules for function names are the same as for variable names: letters, numbers and some punctuation marks are legal, but the first character can't be a number. You can't use a

keyword as the name of a function, and you should avoid having a variable and a function with the same name.

The empty parentheses `()` after the name, `print_lyrics`, indicate that this function doesn't take any arguments. Later we will build functions that take arguments as their inputs. The first line of the function definition is called the *header*; the rest is called the body. The header has to end with a colon `:` and the body has to be indented. By convention, the indentation is always four spaces. The body can contain any number of statements. Usually when you press enter on Jupyter Notebook or any good text-editor it will automatically indent it for you.

Let's see what kind of object is `print_lyrics` is

[98]:
```
type(print_lyrics)
```

[98]:
```
function
```

No surprise! The value of `print_lyrics` is a *function object*, which has type *"function"*.

Now how do we call the function? The syntax for calling the new function is the same as for built-in functions:

[99]:
```
print_lyrics()
```

```
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Now let's create a function with inputs. Here we define a function `add_two_things` with two inputs `a` and `b`.

[100]:
```
def add_two_things(a,b):
    addition=a+b
    print(addition)
```

Now let's call the function. I want to add 3 and 2.

[101]:
```
add_two_things(3,2)
```

```
5
```

We do get the desired result. But if we want to use this function somewhere else like maybe multiply with 3: `add_two_things(3,2)*3`.

[102]:
```
add_two_things(3,2)*3
```

```
5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/01/jybr74m10l5d2ldzyht8kj5h0000gn/T/ipykernel_57896/4170487392.py␣
 ↪in <module>
----> 1 add_two_things(3,2)*3
```

```
TypeError: unsupported operand type(s) for *: 'NoneType' and 'int'
```

You will run into an error. To fix this we need our functin to return a value. So we use the `return` instead of `print()`

[ ]:
```
def add_two_things(a,b):
    addition=a+b
    return addition
```

Now if try to run the same operation: `add_two_things(3,2)*3`. We have no problem!

[ ]:
```
add_two_things(3,2)*3
```

Now let's make this calculator more interactive.

[ ]:
```
def add_two_things(a,b):
    addition=a+b
    return addition
a=int(input('What is the number a?'))
b=int(input('What is the number b?'))
add_two_things(a,b)
```

[ ]:
```

```