Lecture 2 Code Examples

February 4, 2024

1 Types and Expressions

Jupyter Notebooks provide a rich interface to interactive Python. To read more about how to use them, check out our How To Jupyter guide.

1.1 Types in Python

The built-in `type()` function lets us see the data type of various values in Python.

Note: The one line phrases after # are comments, they are ignored during execution.

[1]:
```
type(134)
```

[1]:
```
int
```

[2]:
```
# single quotes
type('134')
```

[2]:
```
str
```

[3]:
```
# double quotes
type("134")
```

[3]:
```
str
```

[4]:
```
type(3.14159)
```

[4]:
```
float
```

[5]:
```
type('')
```

[5]:
```
str
```

[6]:
```
type(0)
```

[6]:
```
int
```

[7]:
```
type(False)
```

[7]:
```
bool
```

[8]:
```
type(True)
```

[8]:
```
bool
```

[9]:
```
type(None)
```

[9]:
```
NoneType
```

[10]:
```
# What will this return?
type(1000 / 3)
```

[10]:
```
float
```

[11]:
```
# How about this?
type(1000 % 3)
```

[11]:
```
int
```

1.2 Simple Expressions using Ints and Floats

The Python interactive interpreter can perform calculations of different expressions just like a calculator.

Let's try some simple arithmetic expressions below.

[12]:
```
11 + 7
```

[12]:
```
18
```

[13]:
```
11 - 7
```

[13]:
```
4
```

[14]:
```
# floating point division
19 / 3
```

[14]:
```
6.333333333333333
```

[15]:
```
# integer division
19 // 3
```

[15]:
```
6
```

[17]:
```
# modulo operator - computes remainder
23 % 5
```

[17]: 3

[18]:
```
# exponentiation
2 ** 3
```

[18]: 8

[19]:
```
3 + 4 * 5
```

[19]: 23

[20]:
```
# parentheses can be used to override
# operator precedence (or order of operations)
(3 + 4) * 5
```

[20]: 35

1.2.1 Simple Expressions Using Strings

A string is a sequence of characters that we write between a pair of double quotes or a pair of single quotes.

[21]:
```
# the string is within double quotes
"Happy"
```

[21]: 'Happy'

[22]:
```
# we can also use single quotes, it is still a string
'Birthday!'
```

[22]: 'Birthday!'

[23]:
```
# example of string concatenation
"Happy" + 'Birthday'
```

[23]: 'HappyBirthday'

[24]:
```
# space is a character and matters in strings
"Happy " + 'Birthday'
```

[24]: 'Happy Birthday'

String concatenation is the operation of chaining two or more strings together to form one string.

Notice that the operator '+' behaves differently depending on the the **type** of its operands.
* When + is applied to ints or floats, it adds them mathematically. * When + is applied to strings, it concatenates them.

What if we apply + to a combination of int and str? Guess what will happen below:

[25]:
```
"CS" + 134
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[25], line 1
----> 1 "CS" + 134

TypeError: can only concatenate str (not "int") to str
```

This results in a `TypeError`, which happens when an operator is given operand values with types (e.g. `int`, `float`, `str`) that do not correspond to the expected type. You cannot add a string to an integer. That does not make sense.

How can we fix the expression, so that it no longer leads to an error?

[26]:
```
"CS " + "134"
```

[26]:
```
'CS 134'
```

Multiplication operator on strings: What do you think will happens if we do this:

[27]:
```
'*Williams*' * 3
```

[27]:
```
'*Williams**Williams**Williams*'
```

[28]:
```
# will this work?
'*Williams*' * 2.0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[28], line 2
      1 # will this work?
----> 2 '*Williams*' * 2.0

TypeError: can't multiply sequence by non-int of type 'float'
```

[29]:
```
# will this work?
'x' * 'y'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 2
      1 # will this work?
----> 2 'x' * 'y'

TypeError: can't multiply sequence by non-int of type 'str'
```

Summary: The operators + and * are the only ones you can use with values of type string. Both of these operators generate concatenated strings. Be careful when using the * operator. One of the operands needs to be an integer value.

1.2.2 Variables and Assignment

Variables are used to give names to values using the assignment operator (=). A variable is essentially a placeholder for a stored value that you want to reuse or update later in the program.

Important: To avoid confusion, the symbol = is sometimes referred to as "gets" or "is assigned to", but not "equals"!

[30]:
```
# what does this do
num = 23
```

Notice: The above statement did not produce an output (that is, no Out[] cell). The statement defines the variable num and assigns it the value 23.

[31]:
```
# once a variable is defined, we can "refer" to it
# and ask for its current value
num
```

[31]: 23

[32]:
```
# to evaluate this expression, python first replaces num with its value and␣
↪then performs the operation
num * 2
```

[32]: 46

[33]:
```
# updating value of num (right evaluated first, and new
# value assigned to variable on left)
# same as num = num + 1
num *= 2
```

[34]:
```
# what will be printed?
print(num)
```

```
46
```

1.3 Built-in Functions: `print()`, `input()`, `int()`

Python comes with a ton of built-in capabilities. In this section we will explore some useful built-in functions.

1.3.1 print()

The `print()` function is used to **display** characters on the screen. When we print something, notice that there is no output (no Out[] cell). This is because it does not **return** a value as an

output (like a Python expression). It simply performs an action. (We'll revisit what it means to "return" a value soon.)

[42]:
```
print('Welcome to Computer Science')
```

```
Welcome to Computer Science
```

Also notice there are no quotation arounds the printed text. Technically it is not a string!

[43]:
```
print(50 % 4)  # we can put Python expressions within a print statement directly
```

```
2
```

[44]:
```
print('****' + '####' + '$$$$')
```

```
********$$$$
```

When `print()` is called with multiple arguments, it prints them all, separated by spaces. (*Arguments* are the values inside of the parentheses, separated by commas.)

[45]:
```
print('hakuna', 'matata', 24*7)
```

```
hakuna matata 168
```

1.3.2 input()

The `input()` function is used to take input from the user. We specify the prompt we want the user to see within the parentheses. By default, input values are always of type string.

[47]:
```
# the string within the parens is the prompt the
# user will see in the terminal
input('Enter your name')
```

```
Enter your name Shikha
```

[47]:
```
'Shikha'
```

Notice: Unlike the `print()` function, `input()` returns a value (which is just the input entered by user).

[48]:
```
# input with a colon and space, sometimes easier
# for user to understand in terminal
input('Enter your age: ')
```

```
Enter your age:  12
```

[48]:
```
'12'
```

Notice: Notice anything about the type of the input value?

1.3.3 int()

The `int()` function is used to convert strings of digits to integers.

[49]:
```
age = input('Enter your age: ')
```

```
Enter your age:  11
```

[50]:
```
age
```

[50]:
```
'11'
```

[51]:
```
type(age)
```

[51]:
```
str
```

[52]:
```
int(age)
```

[52]:
```
11
```

[53]:
```
# will this work?
int('45.78')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[53], line 2
      1 # will this work?
----> 2 int('45.78')

ValueError: invalid literal for int() with base 10: '45.78'
```

[54]:
```
pi = 3.14159
```

[55]:
```
# what does this return?
int(pi)
```

[55]:
```
3
```

Summary of `int()` * When given a string that is a sequence of digits (optionally preceded by +/-), the int() function returns the corresponding integer; on any other string it results in a ValueError * When given a float value, the int() function returns the integer formed by truncating it *towards* zero * When given an integer, the int() function returns that same integer

1.4 Putting it all together

Now that we know how to take user input, store it, use it to compute simple expressions, and print values, we are ready to put it all to good use.

Let us write a short Python program that:

- take as input the current age of the user
- computes the year they were born (assuming current year is 2022)
- print the result

[59]:
```
age = int(input("Enter your age: "))
print("Year you were born is: ", 2022-age)
```

```
Enter your age:  11
```

```
Year you were born is:  2011
```