Variables And Operators

April 21, 2020

Variables, Data Types and Operators

1 Working with Python - Interactive Mode

- Introducing IDLE **Shell**
- Each session is an interactive instance of Python interpreter
- When quit from the session, the definitions made (functions and variables) are lost.

2 Explore basic data types

Let's try some simple Python commands to explore some basic data types: - Numbers (int, float) - Strings - Lists

3 Numbers

The Python interpreter can act as a simple calculator - Expression syntax is straightforward: the operators `+`, `-`, `*` and `/` work just like math. - Parentheses `()` can be used for grouping and order or precedence.

[21]:
```
2 + 2
```

[21]:

[6]:
```
50 - 5*6
```

[6]:
```
20
```

[7]:
```
(50 - 5)*6
```

[7]:
```
270
```

4 Numbers - `int`, `float` and beyond

- The integer numbers (e.g. `2`, , `20`) have type `int`
- The ones with a fractional part (e.g. `5.0`, `1.6`) have type `float`.
- In addition to `int` and `float`, Python supports other types of numbers, such as `Decimal` and `Fraction`.

- Python also has built-in support for complex numbers, and uses the `j` or `J` suffix to indicate the imaginary part (e.g. `3+5j`).

5 Numbers - Division

- Division (/) always returns a `float`.
- To do floor division and get an integer result (discarding any fractional result), use the `//` operator;
- To calculate the remainder, use `%`

[10]:
```
17 / 3  # Classic division returns a float
```

[10]:
```
5.666666666666667
```

[12]:
```
17 // 3  # Floor division discards the fractional part
```

[12]:
```
5
```

[13]:
```
17 % 3  # The % operator returns the remainder of the division
```

[13]:
```
2
```

[ ]:
```
5 * 3 + 2  # result * divisor + remainder
```

6 Numbers - Power

- Use the `**` operator to calculate powers:
- `**` has higher precedence than negative sign `-`; if you want a negative base, use parentheses

[15]:
```
2 ** 7  # 2 to the power of 7
```

[15]:
```
128
```

[16]:
```
-3**2  # Same as -(3**2)
```

[16]:
```
-9
```

[17]:
```
(-3)**2
```

[17]:
```
9
```

7 Numbers - Mixed type conversion

- Python provides full support for floating point numbers; operators with mixed type operands convert the integer operand to floating point

[19]:
```
3 * 3.75 / 1.5
```

[19]:
```
7.5
```

[20]:
```
7.0 / 2
```

[20]:
```
3.5
```

8 Variables

- A **literal** is a fixed value
- A variable is a name given to a **memory** location in a computer, where a value can be stored; A variable can be used in place of a literal
- Variables names start with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0-9).
- Python is **case sensitive**. X and x are treated as different variables.
- The equal sign (=) assigns a value to a variable

[22]:
```
width = 20
height = 5 * 90
width * height
```

[22]:
```
9000
```

9 Variables - Define

- A variable is defined when first assigned a value
- A variable must be defined before it can be used. If a variable is not "defined" (assigned a value), using it produces an error.
- When working in interactive mode, the variables definitions made are lost if quit from shell.

[23]:
```
n  # Try to access an undefined variable
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-23-2d383632fd8e> in <module>
----> 1 n  # Try to access an undefined variable.


NameError: name 'n' is not defined
```

10 Student Exercise

- Calculate paid price for an item worth of $100 with tax rate 12.5%

[24]:
```
tax = 12.5 / 100
price = 100
```

```
price += price * tax # condensed form for price = price + price * tax
price
```

[24]:
```
112.5
```

11 Strings

- A string is a series of characters.
- Strings can enclosed in single quotes (`'...'`) or double quotes (`"..."`).
- Use \ to escape quotes, that is, to use a quote within the string itself.

[ ]:
```
'spam eggs'  # Single quotes
```

[ ]:
```
'doesn\'t'  # Use \' to escape the single quote
```

[ ]:
```
"doesn't"  #    or use double quotes instead
```

12 Student Exercise

- Compose a string for `"Yes," he said, "Isn't it?"`

[25]:
```
'"Yes," he said, "Isn\'t it?"'
```

[25]:
```
'"Yes," he said, "Isn\'t it?"'
```

[26]:
```
"\"Yes,\" he said, \"Isn\'t it?\""
```

[26]:
```
'"Yes," he said, "Isn\'t it?"'
```

13 Strings - `print()`

- `print()` is a **built-in function** to display the value of a variable
- `print()` produces a more readable output by omitting quote signs and printing escaped and special characters (prefaced by \, such as `\n`, `\t`, etc)

[27]:
```
s = 'First line.\nSecond line.'  # \n means newline
s  #  ithout print(), \n is included in the output
```

[27]:
```
'First line.\nSecond line.'
```

[28]:
```
print(s)  #  ith print(), \n produces a new line
```

```
First line.
Second line.
```

14 Student Excerise

If you don't want escaped characters (prefaced by \) to be interpreted as special characters - use *raw strings* by adding an `r` before the first quote, or, - use another \ before espcaped characters

[29]:
```
print('C:\some\name')  #  ere \n means newline!
```

```
C:\some
ame
```

[30]:
```
print('C:\\some\\name')  # Note escape \ with another \
```

```
C:\some\name
```

[31]:
```
print(r'C:\some\name')  # Note the r before the quote
```

```
C:\some\name
```

15 Strings - Multi-line

- String literals can span multiple lines by triple-quotes: """...""" or '''...'''.

[ ]:
```
print("""Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to""")
```

16 Strings - +, - *Strings can be* concatenated* (glued together) with the + operator, or

- Repeated with *

[32]:
```
# 3 times 'un', followed by 'ium'
3 * 'un' + 'ium'
```

[32]:
```
'unununium'
```

[33]:
```
prefix = 'Py'
word = prefix + 'thon'
word
```

[33]:
```
'Python'
```

17 String - Indexing

- A string can be *indexed* (subscripted) to extract individual character, with the first character having index 0 and the last character as -1

+—+—+—+—+—+—+—+ | P | y | t | h | o | n | +—+—+—+—+—+—+—+ 0 1 2 3 4 5 6 -6 -5 -4 -3 -2 -1

[34]:
```
word[0]  # Character in position 0
```

[34]:
```
'P'
```

[35]:
```
word[-1]  # Last character
```

[35]:
```
'n'
```

[1]:
```
word[ 2]  # The word only has 6 characters
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-1-e89 f93573ea> in <module>
----> 1 word[ 2]  # The word only has 6 characters.


NameError: name 'word' is not defined
```

18 Strings - Slicing

- *Slicing* extracts a substring.
- To slice, use a *range* `start:end`, where the `start` is included but the `end` is excluded
- If either position omitted, the default `start` is 0 and the default `end` is the length of the string

[36]:
```
word[0:2]  # Characters from position 0 (included) to 2 (excluded)
```

[36]:
```
'Py'
```

[37]:
```
word[:2]   # Character from the beginning to position 2 (excluded)
```

[37]:
```
'Py'
```

19 Student Excercise

Use slicing to extract `'on'` from `'Python'`

[38]:
```
word[ :6]  # Characters from position 4 (included) to 6
```

[38]:
```
'on'
```

[39]:
```
word[ :]  # Characters from position 4 (included) to the end
```

[39]:
```
'on'
```

[40]:
```
word[-2:] # Characters from the second-last (included) to the end
```

[40]:
```
'on'
```

20 Student Exercise

What is `s[:i] + s[i:]`? For example `word[:2] + word[2:]` or `word[: ] + word[ :]`

[41]:
```
word[:2] + word[2:]
```

[41]:
```
'Python'
```

[42]:
```
word[: ] + word[ :]
```

[42]:
```
'Python'
```

21 Strings - Immutable

Python strings are immutable, which means they cannot be changed. Therefore, assigning a value to an indexed position in a string results in an error:

[43]:
```
word[0] = 'J'
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input- 3-91a956888ca7> in <module>
----> 1 word[0] = 'J'


TypeError: 'str' object does not support item assignment
```

22 String - `Len()`

- The **built-in function** `len()` returns the length of a string

[44]:
```
s = 'supercalifragilisticexpialidocious'
len(s)
```

[44]:
```
3
```

23 Lists

- *List* is a **compound** data types, which are used to group together other values.
- A List can be written as a sequence of **comma-separated** values (items) between square brackets as `[]`.
- Lists might contain items of different types, but usually the items all have the same type.

[45]:
```
squares = [1,  , 9, 16, 25]
squares
```

[45]:
```
[1,  , 9, 16, 25]
```

24 List - indexing, slicing, +

- Like strings, lists can be indexed and sliced
- Lists also support concatenation with the + operator

[46]:
```
squares[0]  # Indexing returns the item
```

[46]:
```
1
```

[47]:
```
squares[-3:]  # Slicing returns a new list
```

[47]:
```
[9, 16, 25]
```

[48]:
```
squares[:]
```

[48]:
```
[1,  , 9, 16, 25]
```

[49]:
```
squares + [36,  9, 6 , 81, 100]
```

[49]:
```
[1,  , 9, 16, 25, 36,  9, 6 , 81, 100]
```

25 List - Mutable

- Unlike strings, which are immutable, lists are a mutable type, which means you can change any value in the list

[50]:
```
cubes = [1, 8, 27, 65, 125]  # Something's wrong here
  ** 3  # the cube of 4 is 64, not 65!
```

[50]:
```
6
```

[51]:
```
cubes[3] = 6   # Replace the wrong value
cubes
```

[51]:
```
[1, 8, 27, 6 , 125]
```

26 List - append()

Use the list's `append()` method to add new items to the end of the list

[52]:
```
cubes.append(216)  # Add the cube of 6
cubes.append(7 ** 3)  # and the cube of 7
cubes
```

[52]:
```
[1, 8, 27, 6 , 125, 216, 3 3]
```

27 List - Versatile Manipulation

[53]:
```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
letters
```

[53]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

[54]:
```
letters[2:5] = ['C', 'D', 'E']# Replace some values
letters
```

[54]:
```
['a', 'b', 'C', 'D', 'E', 'f', 'g']
```

[55]:
```
letters[2:5] = []# Now remove them
letters
```

[55]:
```
['a', 'b', 'f', 'g']
```

[56]:
```
# Clear the list by replacing all the elements with an empty list
letters[:] = []
letters
```

[56]:
```
[]
```

28 List - `Len()`

- The method `len()` function applies to lists to get the size of a list.

[57]:
```
letters = ['a', 'b', 'c', 'd']
len(letters)
```

[57]:

29 List - Nested

- You can nest lists, which means to create lists that contain other lists.

[58]:
```
a = ['a', 'b', 'c']
n = [1, 2, 3]
x = [a, n]
x
```

[58]:
```
[['a', 'b', 'c'], [1, 2, 3]]
```

[59]:
```
x[0]
```

[59]:
```
['a', 'b', 'c']
```

[60]:
```
x[0][1]
```

[60]:
```
'b'
```