Ch02-Data-Variables

August 7, 2020

1 2. Values, expressions and statements

- http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html

1.1 Topics

- data values and types
- variables
- expressions and statements
- numeric operators and basic computations
- order of operations
- standard input and output
- type casting
- Composition and algorithm

1.2 2.1 Values and data types

- a value is one of the fundamental things or data – like a letter or number – that a program manipulates
- these values are clssified into types
- Python fundamentally supports two major data types: numbers and strings
- Boolean (True/False) is also supported type

1.2.1 Numbers

- Integer (int)
  - +/- whole numbers: 199, -98, 0
- Float
  - +/- real numbers - numbers with decimal points: 99.99, -0.01

1.2.2 Strings

- Python uses **str** abbreviation for String type
- strings are one or more characters represent using single, double or tripple quotes

1.3 2.2 First program - hello world!

[1]:
```
#-----------------------------------------------------------
# hello world program
# by: John Doe
# Jan 1, 2017
# Copyright: Anyone may freely copy or modify this program
#-----------------------------------------------------------
print('hello world!') # say hello to the beautiful world!
# In python 2: print is a statement not a function
# print 'Hello World!'
```

```
hello world!
```

1.4 2.3 Comments

- Programming languages provide a way to write comment along with the codes
- Python uses # symbol to write a single line comment
- comments are for humans/programmers to convey/explain what the code is doing or supposed to do
- Python interpreter ignores the comments

1.5 2.4 Data Types

- built-in type() function can be used to know type of data
- functions will be covered in Chapter 04

[2]:
```
type(100)
```

[2]:
```
int
```

[3]:
```
type(-9)
```

[3]:
```
int
```

[4]:
```
type(1000.99345435)
```

[4]:
```
float
```

[5]:
```
type(-2.345)
```

[5]:
```
float
```

[6]:
```
type('Hello World!')
```

[6]:
```
str
```

[7]:
```
type('A')
```

```
[7]: str
```

```
[8]: print("hello")
```

```
hello
```

```
[9]: type("17")
```

```
[9]: str
```

```
[10]: type("""Triple double quoted data""")
```

```
[10]: str
```

```
[11]: type('''Type of Triple single quote data is''')
```

```
[11]: str
```

```
[12]: a = "hello"
```

```
[13]: a
```

```
[13]: 'hello'
```

1.6 2.5 Type conversion/casting

- data needs to be converted from one type to another as needed
- the process is called type casting
- use built-in functions such as str(), int() and float()
- **str(value)** - converts any value into string
- **int(value)** - converts numeric value into int
- **int(value)** - converts numeric value into float

```
[14]: data = 'hello' # can't conver it to int or float types
```

```
[15]: type(data)
```

```
[15]: str
```

```
[16]: data = '100'
      type(data)
```

```
[16]: str
```

```
[17]: data
```

```
[17]: '100'
```

[18]:
```
num = int(data)
type(num)
```

[18]:
```
int
```

[19]:
```
num
```

[19]:
```
100
```

[20]:
```
price = float('500.99')
type(price)
```

[20]:
```
float
```

[21]:
```
float('hi')
```

```
        ␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-21-4db454aaf92e> in <module>
   ----> 1 float('hi')


       ValueError: could not convert string to float: 'hi'
```

[22]:
```
num = 99.99
strNum = str(num)
type(strNum)
```

[22]:
```
str
```

[23]:
```
type(True)
```

[23]:
```
bool
```

[24]:
```
type(False)
```

[24]:
```
bool
```

1.7 2.6 Statements

- a **statement** is an instruction that the Python interpreter can execute

- we've seen assignment statements so far
- we'll later explore for, if, import, while and other statements

1.8 2.7 Expressions

- an **expression** is a combination of values, variables, operators, and calls to functions
- expressions are evaluated giving a results

```
[25]: 1+2
```

```
[25]: 3
```

```
[26]: len('hello')
```

```
[26]: 5
```

```
[27]: print(2+3*4)
```

```
14
```

1.9 2.8 Standard Output

- printing or writing output to common/standard output such as monitor
- way to display the results and interact with the users of your program
- use print() function

```
[28]: print('''"Oh no", she exclaimed, "Ben's bike is broken!"''')
```

```
"Oh no", she exclaimed, "Ben's bike is broken!"
```

```
[29]: print(34.55)
```

```
34.55
```

```
[30]: print(2+2)
```

```
4
```

1.10 2.9 Escape Sequences

- some letters or sequence of letters have special meaning to Python
- single, double and tripple single or double quotes represent string data
- use backslash \ to represent these escape sequences, e.g.,
  - \n - new line
  - \\ - back slash
  - \t - tab
  - \r - carriage return
  - \' - single quote
  - \" - double quote

[31]:
```
print('What\'s up\n Shaq O\'Neal?')
```

```
What's up
 Shaq O'Neal?
```

[32]:
```
print('hello \there...\n how are you?')
```

```
hello   here…
 how are you?
```

[33]:
```
print('how many back slahses will be printeted? \\\\')
```

```
how many back slahses will be printeted? \\
```

[34]:
```
print(r'how many back slahses will be printeted? \\\\')
```

```
how many back slahses will be printeted? \\\\
```

1.11 2.10 Variables

- variables are identifiers that are used to store values which can be then easily manipulated
- variables give names to data so the data can be easily referenced by their names over and again
- rules and best practices for creating identifiers and variable names:
  - can't be a keyword – what are the built-in keywords?
  - can start with only alphabets or underscore (_)
  - can't have symbols such as $, %, &, white space, etc.
  - can't start with a digit but digits can be used anywhere else in the name
  - use camelCase names or use _ for_multi_word_names
  - use concise yet meaningul and unambigous names for less typing and avoid typos

[35]:
```
help('keywords')
```

```

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

[36]:
```
# variable must be defined/declared before you can use
print(x)
```

```
       ␣
 ↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
 ↪last)

       <ipython-input-36-293ab258265c> in <module>
         1 # variable must be defined/declared before you can use
   ----> 2 print(x)


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

[37]:
```
x = 'some value'
```

[38]:
```
print(x)
```

```
some value
```

[39]:
```
# Exercise: Define a bunch of variables to store some values of different types
var1 = -100
num = -99.99
name = 'John'
lName = 'Smith'
MI = 'A'
grade = 10.5
Name = 'Jake'
grade = 19.9
```

1.12 2.11 Dynamic Typing

- type of variables are dynamically evaluated in Python when the assignment statement is executed
- same variable can be uesd to hold different data types

[40]:
```
var = 100
```

[41]:
```
var = 'hello'
```

[42]:
```
var = 99.89
```

[43]:
```
var
```

[43]: 99.89

1.13 2.12 Visualize variable assignments in pythontutor.com

Click Here

1.14 2.13 Computation - operators and operands

- **operators** are special tokens/symbols that represent computations like addition, multiplication and division
- the values an operator uses are called **operands**
- some binary operators that take two operands
  - addition: 10 + 20
  - subtraction: 20 - 10
  - true division: 10 / 3
  - multiplication: 7 * 9
  - integer division: 10 // 3
  - remainder or modulus operator: 10 % 2
  - power: 2 ** 3

[44]:
```
# Exercise: play with some examples of various operators supported by Python
```

1.15 2.14 Order of operations

- depends on the rules of precedence

uses PEMDAS rule from high to low order

1. Parenthesis
2. Exponentiation
3. Multiplication and Division (left to right)
4. Addition and Subtraction (left to right)

[45]:
```
# some examples
print(2 * 3 ** 2)
```

```
18
```

[46]:
```
x = 1
y = 2
z = 3
ans = x+y-z*y**y
print(ans)
```

```
-9
```

1.16 2.15 Operations on strings

- + and * operators also work on strings
- Chapter 08 covers more on string data type and operations

```
[47]: # some examples
      fname = "John"
      lname = "Smith"
      fullName = fname + lname
      print(fullName)
```

```
JohnSmith
```

```
[48]: gene = "AGT"*10
```

```
[49]: print(gene)
```

```
AGTAGTAGTAGTAGTAGTAGTAGTAGTAGT
```

1.17 2.16 Standard input

- read data from standard or common input such as keyboards
- allows your program to receive data during program execution facilitating user interactions
- input values will have type string even if numbers are entered
- use variables to store the data read from standard input

```
[50]: name = input('What is your name? ')
```

```
What is your name?
```

```
[51]: print('hello,', name)
```

```
hello,
```

```
[52]: num = input('Enter a number =>')
      print('You entered: ', num)
      print('type of', num, '=', type(num))
```

```
Enter a number =>
You entered:
type of  = <class 'str'>
```

```
[53]: num
```

```
[53]: ''
```

```
[54]: # str must be casted into int or float depending on the value required
      num = int(num)
      print('type of', num, '=', type(num))
```

```
       ␣
↪---------------------------------------------------------------------------
```

```
        ValueError                                Traceback (most recent call␣
↪last)

        <ipython-input-54-1935f54633b6> in <module>
          1 # str must be casted into int or float depending on the value␣
↪required
    ----> 2 num = int(num)
          3 print('type of', num, '=', type(num))


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

1.18 2.17 Composition

- break a problem into many smaller sub-problems or steps using high-level algorithm steps
- incrementally build the solution using the sub-problems or steps

1.19 2.18 Algorithm

- step by step process to solve a given problem
  - like a recipe for a food menu
- what are the steps you'd give a martian to buy grocery on earth?
  1. Make a shopping list
  - Drive to a grocery store
  - Park your car
  - Find items in the list
  - Checkout
  - Load grocery
  - Drive home

1.20 2.19 Exercises

1.20.1 1. area and perimeter of a rectangle

- write a python script that calculates area and perimeter of a rectangle

[55]:
```
# Demonstrate composition step by step
# Algorithm steps
# 1. Get length and width of a rectangle
#     a. hard coded values OR
#     b. prompt user to enter length and width values
#         i. convert length and width into right data types
# 2. Find area = length x width
# 3. Find perimeter = 2(length + width)
# 4. Display calculated results
```

1.20.2 2. area and circumference of a circle

- write a python script that calculates area and circumference of a circle
- area of a circle is calculated using equation: πr^2
- perimeter of a circel is calculated using equation: $2\pi r$
- you can use $\pi = 3.14159$

[56]:
```
# Demonstrate composition step by step
# Algorithm steps
```

1.20.3 3. Body Mass Index (BMI)

- write a python script that calculates BMI of a person
- BMI is body mass in kg divided by the square of body height in meter with the units of kg/m^2
- https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmicalc.htm

1.20.4 4. area and perimeter of a triangle

- write a python script that finds area and perimeter of a triangle given three sides
- Hint: Google and use Heron's formula to find area of triangle

[ ]:
```

```