2.Basic_data_types

November 29, 2018

1 Basic Data Types

1.1 Numbers

- integer - 3, 0, 7, -10, 42 -> int
- fraction - 0.0, 3.5, 2e-6 -> float
- complex - 3 + 4i -> special type in math library

1.2 Number Representation Problems

- binary numeral system
- limited disk space

```
In [25]: 1 / 3

Out[25]: 0.3333333333333333

In [26]: 0.1 / 3

Out[26]: 0.03333333333333333

In [27]: 0.01 / 3

Out[27]: 0.0033333333333333335

In [28]: 0.0033333333333333335 * 3

Out[28]: 0.01
```

1.3 Arithmetic operations

- Addition - +
- Subtraction - −
- Multiplication - ∗
- Exponentiation - ∗∗
- Division - /
- Integer (floor) division - //
- Modulo (remainder) - %

```
In [29]: 3 + 4
Out[29]: 7
In [30]: 10 - 2
Out[30]: 8
In [31]: 5 * 5
Out[31]: 25
In [32]: -3 / 2
Out[32]: -1.5
In [33]: 2 ** 3
Out[33]: 8
In [34]: 2.0 ** 3
Out[34]: 8.0
In [35]: 2 ** 3.0
Out[35]: 8.0
In [36]: 2 ** 0.5
Out[36]: 1.4142135623730951
In [37]: 10 // 3
Out[37]: 3
In [38]: -10 // 3
Out[38]: -4
In [39]: 10 % 2
Out[39]: 0
In [40]: 11 % 2
Out[40]: 1
In [41]: 11 % 3
Out[41]: 2
In [42]: -11 % 3
Out[42]: 1
```

$a = (a \ // \ b)$ $ * b + a$ $\% \ b$

1.4 Strings

Any text can be a string
* "a" * "I'm a string!" * "0.5"

In python strings should be enclosed in quotes, double quotes or their triplication double quotes
* 'd' * "42" * """Massive string""" * '''Another massive string'''

```
In [43]: "a"

Out[43]: 'a'

In [44]: 'string'

Out[44]: 'string'

In [45]: """multiline
         string"""

Out[45]: 'multiline\nstring'

In [46]: "jvd
         vfd
         vfdkm"


          File "<ipython-input-46-703b84314c3a>", line 1
        "jvd
           ^
    SyntaxError: EOL while scanning string literal
```

1.5 Print strings

`print()` is a function to pass something to display as a string
Something should be placed in parenthesis

```
In [ ]: 'Hello, guys!'

In [48]: print('Hello, guys!')

Hello, guys!
```

1.6 How to print quote inside string?

How would you print this phrase?
There are several types of quotes in language - ', "" and even their multiplication - """", '''

```
In [49]: print("There are several types of quotes in language - '', "" and even their multiplica
```

```
  File "<ipython-input-49-aa62a2043678>", line 1
    print("There are several types of quotes in language - '', "" and even their multiplicat

SyntaxError: EOF while scanning triple-quoted string literal
```

Escape character is really for such case - Backslash is a special character which disable meaning of other special characters (quotes in this case)

```
In [50]: print("There are several types of quotes in language - '', \"\" and even their multipli

There are several types of quotes in language - '', "" and even their multiplication - """""", '
```

1.7 Another function to look at strings

`repr()` will show how something is represented as string internally
Again something should be placed inside parenthesis (it is common for functions)

```
In [51]: repr('a')

Out[51]: "'a'"

In [52]: repr(3)

Out[52]: '3'

In [53]: print("There are several quotes in this sentence - \'\', \"\" and a backslash itself \\

There are several quotes in this sentence - '', "" and a backslash itself \


In [54]: repr("There are several quotes in this sentence - \'\', \"\" and a backslash itself \\"

Out[54]: '\'There are several quotes in this sentence - \\\'\\\', "" and a backslash itself \\\\

In [55]: "There are several types of quotes in language - '', \"\" and even their multiplication

Out[55]: 'There are several types of quotes in language - \'\', "" and even their multiplication
```

1.8 Boolean type

False and True values which means ... well False and True)
Internally encoded as 0 and 1

1.9 Comparisons

Typical actions with numbers: * Greater - > * Greater or equal - >= * Lesser - < * Lesser or equal - <= * Equal - == * Not equal - !=

Yields a boolean value

```
In [56]: 5 > 3
```

```
Out[56]: True
```

```
In [57]: 12 < 3
```

```
Out[57]: False
```

```
In [58]: 6 >= 5.5
```

```
Out[58]: True
```

```
In [59]: 6 >= 6
```

```
Out[59]: True
```

```
In [60]: 3 == 3
```

```
Out[60]: True
```

```
In [61]: 6 == -2
```

```
Out[61]: False
```

```
In [62]: 2.0 != 3.0
```

```
Out[62]: True
```

```
In [63]: 0 != 0.0
```

```
Out[63]: False
```

1.10 About boolean values

```
In [64]: False == 0
```

```
Out[64]: True
```

```
In [65]: False + 1
```

```
Out[65]: 1
```

```
In [66]: True * 2
```

```
Out[66]: 2
```

1.11 Comparison are not just for numbers

They are for everything that can be compare between themself
Equality (==, !=) works for almost all objects

```
In [67]: 'Ann' < 'Bob'

Out[67]: True

In [68]: 'Ann' == 'Ann'

Out[68]: True

In [69]: 'Ann' == 'ANN'

Out[69]: False
```

1.12 Boolean operations

Yields boolean values * not * or * and

```
In [70]: not True

Out[70]: False

In [71]: False or True

Out[71]: True

In [72]: True and True

Out[72]: True
```

More complex examples

```
In [73]: not False and False or True

Out[73]: True
```

It is better to use parenthesis in complex examples

```
In [74]: (not False) and (False or True)

Out[74]: True

In [75]: not ((False and False) or True)

Out[75]: False
```

1.13 Variables

Containers for values, we can think about them as of parameter or well some variable
Assignment is an act of binding variable with some value
variable assignment_operator value

```
In [76]: a = 5
```

a - variable
= - assignment_operator
5 - value

- Variable name always going on the left
- Value always going on the right

1.14 Why use it?

As was said variables are reusable containers for values which give meaning to values
Imagine converting Celsium degrees to Fahrenheit. First we convert 20 degrees

```
In [77]: 20 * 9 / 5 + 32
```

```
Out[77]: 68.0
```

And after that we wanna convert 232.8 degrees

```
In [78]: 232.8 * 9 / 5 + 32
```

```
Out[78]: 451.0400000000001
```

With variables it will look something like this

```
In [79]: celsium_temperature = 20
         fahrenheit_temperature = celsium_temperature * 9 / 5 + 32
```

```
In [80]: celsium_temperature = 232.8
         fahrenheit_temperature = celsium_temperature * 9 / 5 + 32
```

1.15 Naming rules

- should start with letter or _
- other positions can include digits

1.16 Naming conventions

- use lowercase letters in most cases
- separate words by _
- use meaningful names - it is very important

1.17 Control Flow (branching)

Often you need your program to choose its behaviour depending on situation
Simple example
print whether the number is positive or negative

1.17.1 Small flood

is used to denote commentaries in python - parts of program written for programmers
Comments doesn't processed by interpreter and serve as an explanation of program
Also triple quoted strings are multiline comments

```
In [81]: # I'm a line comment

In [82]: # I'm a long one
         # line comment

In [83]: """
         I'm a multi
         line
         comment
         """

Out[83]: "\nI'm a multi\nline\ncomment\n"

In [84]: number = 5
         # Choose branch according to number value
         if number > 0:
             print("positive")
         if number < 0:
             print("negative")

positive
```

1.18 Morphology of if statement

```
if predicate:
    branch_body
continuation_of_program
```

1.19 Parts of if statement

- if - keyword
- predicate - some boolean expression (at the end it evaluates to True or False)
- : - important colon
- indent - tabulation to denote body of this branch
- removing indent - exit from branch, continue with common part of program

1.20 Types of branching

To starting branching you should use `if`
It is possible to use just one condition

```
temp = -20

if temp < 15:
    print('It\'s cold!')
```

1.21 2 conditions

To make 2 exclusive conditions you should use `else` keyword
If no branches are appropriate, `else` will be executed
It doesn't have predicate
Only 1 branch will be executed

```
temp = -20

if temp < 15:
    print('It\'s cold!')
else:
    print('It\'s normal temperature')
```

1.22 2 nonexclusive conditions

With several `if` you have nonexclusive conditions
Several branches can be executed

```
number = 20

if number > 10:
    print('This number is greater than 10!')
if number > 15:
    print('This number is greater than 15!')
```

1.23 Several conditions

`elif` stands for `else if` and is using to express several conditions
Advanced program to print number quality

```
number = 20

if number > 0:
    print('Number is positive')
elif number < 0:
    print('Number is negative')
else:
    print('Number equal to 0')
```

Remember - predicate of each `if` is evaluated while not appropriate branches (`elif`, `else`) are not

1.24 Nested conditions

Our previous program can be written in other way

```
number = 20

if number > 0:
    print('Number is positive')
elif number <= 0:
    if number == 0:
        print('Number equal to 0')
    else:
        print('Number is negative')
```