Lecture 2 - Python 3 Basics

Computer Programming

Robert D. Vincent and Samia Hilal

Marianopolis College

February 2, 2022

Tokens in Python 3

- In this topic, we will learn the kinds of tokens recognized by Python 3.
- Tokens represent the basic vocabulary of a programming language.
- They include numbers, strings, operators, comments, reserved words, and other punctuation.

A Summary of Tokens in Python

- Numbers: `1`, `3.14159`, `10000000`, etc.
- Names: `__next__`, `myVariable`, `Student1`, etc.
- Reserved words: `if`, `else`, `while`, etc.
- Operators: `:`, `+`, `*`, etc.
- Strings: `"Hello, World!"`, `'X'`
- Comments: Any text following a '#' on a line.

Some conventions I will follow

- Stuff you would actually type in IDLE is presented in `fixed-width` font.
- Python Reserved words in `orange`.
- Python Built-in names in `purple`.
- Comments in `red`.
- Strings in `green`.
- These choices mimic IDLE, the integrated development environment for Python we will use.

Resources

- Official website: `http://www.python.org`
- Visual interpreter: `http://pythontutor.com`
- Book: *Introduction to Computation and Programming Using Python* by John V. Guttag.
- Free online books and courses, e.g.:
  - `https://inventwithpython.com/`
  - `https://learnpythonthehardway.org/book/`
  - `http://greenteapress.com/thinkpython2/html/index.html`
  - `https://www.pythonlearn.com/book.php`

Numbers in the computer

- All data inside the computer is in binary.
  - $0_{10} = 000_2$
  - $1_{10} = 001_2$
  - $2_{10} = 010_2$
  - $3_{10} = 011_2$
  - $4_{10} = 100_2$
  - $5_{10} = 101_2$
  - $6_{10} = 110_2$
  - $7_{10} = 111_2$
- However, Python does not force us to use binary when we write numbers.

Numbers in Python 3

- Like most programming languages, Python recognizes several distinct *types* of numbers.
- Actual numbers are called *literal constants*, (often I will write just *literal* or *constant*).
- Each literal denotes a value or an *object* of a type.
- The two most important types are:
  - Integers (`int`)
  - Floating point (`float`)
- They are stored differently!

Integers

- Integers (usually abbreviated `int`) are whole numbers.
- Integer literals are usually just strings of decimal digits.
- No spaces, periods, or commas!
- Negative numbers have a leading `'-'` character.
- Normally decimal, but can be written in hexadecimal, octal, or binary.

Integer examples

`9`	OK
`01`	No! - leading zeros not permitted.
`1,000,000`	No! - Commas are not legal here.
`1 000 000`	No! - Neither are spaces.
`1000000`	OK
`-501`	OK - negative number.
`- 501`	OK - space here is fine.
`+501`	OK - positive number.

Binary and hexadecimal

- You can write binary or hexadecimal integers in Python 3.
- Binary constants start with the prefix 0b
    - 0b1010
    - 0B11111111
    - 0b0
    - 0b1000000001
- Hexadecimal constants start with the prefix 0x
    - 0xa
    - 0xFF
    - 0X0
    - 0x201

Floating point

- Floating point or *real* numbers.
- Abbreviated `float` in Python.
- Either a decimal point *or* an exponent tells Python 3 that this is a floating point number:
  - `3.0, -.9, 15., 0.0001`
  - `2E21` means 2 times 10^{21}
  - `1.5e-10` means 1.5 times 10^{-10}
- Can include a leading sign (`'-'` or `'+'`), zero or more digits, a decimal point, zero or more digits, and an optional exponent.
- The exponent starts with `'e'`, an optional `'+'` or `'-'` sign, then a string of decimal digits.

Integers vs. floats

- Integers:
  - Store whole numbers.
  - Exact.
- Floating point:
  - Store fractional numbers.
  - Use an exponent, which permits a wide range of values.
  - Inexact (approximate).

Floating point examples

`.9`	OK, no leading zero needed
`1.`	OK
`3e`	No! Must give exponent after 'e'
`1.E100`	Capital 'E' is OK
`3.1415e0`	OK
`1e+100`	OK
`2.1e-100`	OK
`.E+5`	No! Must have a digit before 'e'
`-.9e5`	OK
`0e+99`	OK, but still equals 0.0

Operators and punctuation

- Math operators: + - * / % **.
- Comparison/assignment: = > ! <.
- Separators: Period . comma , colon : and semicolon ;
- Quotes: single ' and double "
- Backslash: \
- Paired characters:
  - Parentheses (and).
  - Square *brackets* [and].
  - Curly *braces* { and }.
- Others: at-sign @, circumflex ^, ampersand &, vertical bar |.

Operators and expressions

- Try these basic mathematical operators in IDLE:
  - Addition: `1 + 2`
  - Subtraction: `5 - 7`
  - Multiplication: `0.6 * 10.0`
  - Division: `27 / 3`
  - Exponentiation: `2 ** 4`
  - Remainder: `17 % 5`
- These are examples of Python 3 *expressions* created by combining numeric values with *operators*.
- A Python 3 expression represents a computation that yields a value.

More on division...

- Python 3 offers two different division operators.
- Floor (or “truncating”) division: `//`
    - `10 // 4` will equal `2`
    - `-10 // 4` will equal `-3`
    - `10.0 // 4.0` will equal `2.0`
- Real or float division: `/`
    - `10 / 4` will equal `2.5`
    - `10 / 2` will equal `5.0`
- The % operator gives the remainder of floor division:
    - `10 % 4` will equal 2.
    - `10 % 3` will equal 1.

Compound expressions

- You can combine multiple operators in an expression:
  - `10 - 5 - 1` Equals 4.
  - `2 + 3 + 5` Equals 10.
  - `2 * 3 * 5` Equals 30.
  - `2 + 3 * 5` Equals 17, not 25!
- Most operators are evaluated from left to right.
- Different operators have higher or lower *precedence*, which affects which part of the expression is evaluated first.

Parentheses

- You can place *sub-expressions* inside parentheses to force them to be evaluated in a preferred order.
- Operators inside parentheses are always evaluated before anything outside the parentheses:
  - `10 - (5 - 1)` Equals 6.
  - `(2 + 3) * 5` Equals 25.
  - `2 + (3 * 5)` Equals 17.
  - `15 / (3 + 2)` Equals 3.

Paired characters

- Each of the paired characters always appears together.
- A left parenthesis must be “closed” by a right parenthesis.
- The most recent left-hand character must be paired with the proper right-hand character.

```
[(1]) # Wrong!
[(1)] # Possibly right.
```

Strings

- Strings are used to represent text.
- Variables can hold strings as well as numbers.
- String data appears between either single or double quotes:
  - `"A"`
  - `"Hello, World!"`
  - `'Marianopolis College'`
  - `'' # Empty string`
- The other type of quote can be freely used inside the string:
  - `"It's what's for dinner"`
  - `'He said "I am waiting outside."'`

Strings, continued

- Strings can contain certain *escape sequences* that represent special characters:

```
"Hello, World!\n"
"A\tB\tC"
```

- Here '\n' represents a "newline" character and a '\t' represents a "tab" character.
- The backslash character introduces an escape sequence.
- Two backslashes are needed to represent a single backslash!

Strings, continued

- The backslash can also be used to include a single quote in a single-quoted string:

  ```
  'It isn\'t hard'
  ```

- Or a double quote in a single-quoted string:

  ```
  "\"So long\", he said."
  ```

Triple-quoted strings

- Often strings are only one-line long.
- If we need to create longer strings that span multiple lines, we can use "triple quotes":

```
message = """Select one of the following:
          1) For account inquiries
          2) For dispute resolution
          0) For immediate assistance."""
```

- This defines a variable named `prompt` that contains a four-line string.

ASCII

- As mentioned previously, it's all ones and zeros inside the computer. How is text represented?
- American Standard Code for Information Interchange
  - Abbreviated "ASCII"
  - Now greatly extended by Unicode and UTF-8

Character	**Decimal**	**Character**	**Decimal**
'A'	65	'Z'	90
'a'	97	'z'	122
'0'	48	'1'	49
'9'	57	' '	32

Unary operators

- The operators we have seen so far as *binary* in that they take two operands.
- There are also a few *unary* operators that take a single operand.
- `-x` Negates the value of the expression x.
- `+x` Preserves the value of the expression x.

Results of math operators

- Like many programming languages, Python 3 will *promote* mathematical types in operations.
- If either operand is a floating point number, the result will be a floating point number.
- Single-slash division always returns a floating point number.

```
5 * 3      # Result is integer.
5 * 3.0    # Result is float.
15 / 3     # Result is 5.0!
17.2 // 3 # Result is 5.0!
```

Names (or Variables)

- We use names for various elements in a program.
- The most common use as *variables*.
- Consist of letters, digits, or underscores ('_').
- Cannot begin with a digit!
- Upper and lower case characters are different, e.g. `Celsius` is distinct from `celsius`.

`__next__`	OK
`GetRange`	OK
`____`	OK
`lesson_1`	OK
`2ndElement`	No!
`zip5890`	OK
`_1`	OK

Reserved words

A `reserved word` has a fixed meaning in Python.
They cannot be used as programmer-defined names.

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

Assignment

- We can *assign* a value to a variable using the equals sign ('='):
  `celsius = 10`
- An *assignment statement* in Python typically consists of an name, followed by the equals sign, then some expression.
- After an assignment, the variable now "contains" the value of the expression.
  `fahr = celsius * 9 // 5 + 32 # fahr=50`

Assignment operators

- It is very common to write statements of the form:

  ```
  value = value * 10
  ```

  or

  ```
  index = index + 1
  ```

- All binary operators support a shorthand with *almost* identical meaning:

  ```
  value *= 10
  index += 1
  ```

- The left-hand part is always evaluated once.

Booleans

- Represent a single logical value
    - `True`
    - `False`
- We will encounter Booleans when we start discussing logical operators, loops, and `if` statements.

Comparison operators

Compare the values of two numbers.

```
a < b      # Is 'a' less than 'b'?
a > b      # Greater than.
a <= b     # Less than or equal.
a >= b     # Greater than or equal.
a == b     # 'a' is equal to 'b'.
a != b     # 'a' not equal to 'b'.
```

Result is a Boolean, either `True` or `False`.

Comparison operators

- Can compare between integer and floating point numbers.
- These are used frequently in `if` and `while` tests.
- Be careful when using equality testing in floating point.

```
>>> 1e+9 + 1e-8 == 1e+9
True                   # ?!?!?!
```

- Floating-point numbers are not always exact!
- Better to check for a very small difference.

Chaining comparisons

Comparisons can be used together as in mathematical notation:

```
>>> x = 2
>>> 1 < x < 3
True
>>> 10 < x < 20
False
>>> 3 > x <= 2
True
>>> 2 == x < 4
True
```

Built-in functions

Python supports several “built-in” functions that are always available.

- **print()** Print a list of arguments.
- **input()** Print a message and return the user’s input.
- **type()** Returns the type of an object
- **str()** Convert the argument to a string.
- **int()** Convert the argument to an integer.
- **float()** Convert the argument to a floating-point number.
- **abs()** Take the absolute value of the argument.

Calling a function

- Another type of expression is a *function call*.
- We start a function call by writing the name of the function, followed by an *argument list*.
- The argument list consists of a pair of parentheses surrounding zero or more comma-separated expressions or *arguments*:

```
abs(-1) # One argument.
print(5 / 2, 5 * 2) # Two arguments.
print() # Zero arguments.
```

Using built-in functions

```
>>> x = 1
>>> y = "test"
>>> print(10, y, x + 1)
10 test 2
>>> z = input("How many? ")
How many? 15
>>> z               # input() returns a string
'15'
>>> type(z)
<class 'str'>
>>> z = float(z) # convert string to float
>>> type(z)
<class 'float'>
>>> print(z)
15.0
```