Data Types

Prof.Dr. Bahadır AKTUĞ
BME362 Introduction to Python

**Compiled from sources given in the references.*

Statically vs. Dynamically Typed Languages

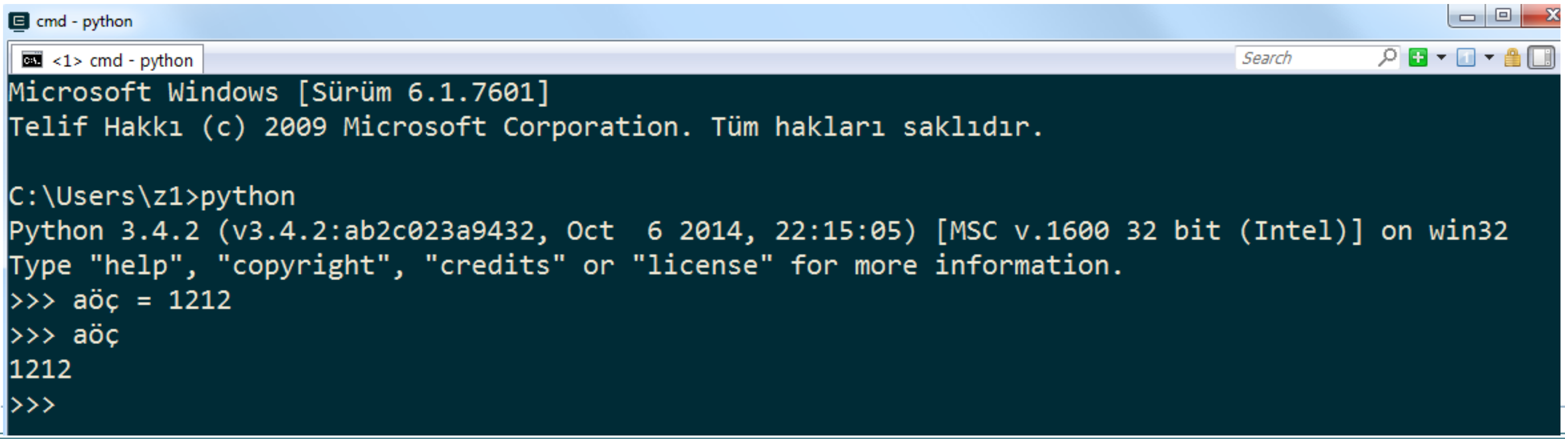
- In statically typed languages, the variables have to be defined before they are used (C/C++/Pascal etc.).
- In statically typed languages, a variable can only have one type that cannot be changed during the program execution.
- In a dynamically typed languages, the variables do not have to be defined before they are assigned.
- In a dynamically typed language, the variables can change their type during the runtime.
- For instance, while variable is an integer at the beginning of a program and then it can be string at the end.
- Python is a dynamicaly typed programming language.

Strongly vs. Weakly Typed Languages

- In strongly typed languages, the operators take the type of each operand into account and a check called "type safety" is applied (C/C++/Pascal etc.).
  - a = "Python"
  - a=1457
  - a = input()
  - print(int(a))
- In a strongly typed language, you cannot add a number to a string or vice versa.
- In a weakly typed language, the usage of the different data types are flexible (Perl, Javascript).
- Python is a strongly typed programming language.

Python Variable Names

- The naming convention with Python 3 has been made quite flexible.
- The variable naming restrictions in Python 3 can be summarized as below:
  - The first character of a variable name must be either a letter (lowercase or uppercase) or "_"
  - The letter could be Unicode
  - Any letter or number can follow after the first character.



```
Microsoft Windows [Sürüm 6.1.7601]
Telif Hakkı (c) 2009 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\z1>python
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> aöç = 1212
>>> aöç
1212
>>>
```

Python Variable Names

- Python is a "case sensitive" language. This also applies to variable as well as commands, functions etc.
- The variable names cannot be chosen from the reserved word list below (they are python commands!)
  - *and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield*

Numbers in Python

Integers

- Decimals (numbers on base 10)
- Octals (numbers in base 8): (they must have "0" and "o")

```
>>> a = 0o20
>>> print(a)
>>> 16
```

- Hexadecimals (numbers on base 16): (they must have "0" and "x/X")

```
>>> a = 0x10
>>> print(a)
>>> 16
```

- Binaries (numbers on base 2): (they must have "0" and "b/B")
- a = 0b110

```
>>> print(a)
>>> 6
```

Conversion to a different base

- Decimal numbers can be converted to other bases:
- From decimal to octal (base 8):

  >>> a = 16

  >>> print(oct(a))

  >>> '0x20' (note that it is converted as a string)
- From decimal to base 16:
- >>> a = 16

  >>> print(hex(a))

  >>> '0x10' (note that it is converted as a string)
- From decimal to base 2:
- >>> a = 16

  >>> print(bin(a))

  >>> '0b10000' (note that it is converted as a string)

Numbers in Python

Integers

- There is no limit for integers:

```
>>> x = 787366098712738903245678234782358292837498729182728
>>> x * x * x
488123970070638215986770162105731315538827586091948617997871122950228891123960901918308618286311523282239313708275589787123005317148968569797875581092352
```

Floating Numbers

```
>>> a = 14.56
>>> a = 2.4583e-8
```

Complex Numbers

- Complex numbers can directly be used in Python.

```
>>> a = 3 – 5j
>>> b = 4 +7j
>>> a+b
```

String type

- There is a need for "string" type to express a sequence of characters (letters, alphanumeric, even numbers, special characters etc).
- ASCII coding allows defining 256 (2^8) different characters.
- However, there are far more letters and symbols than can be accomodated by ASCII. Thus, Unicode standard was established.
- Unicode uses a 4-byte representation instead of ASCII's 1 byte representation of characters.
- 4-byte representation of Unicode allows $(2^8)^4$ > 4 million different characters.
- Since Unicode's 4 byte representation (character mapping) allocates 4-bytes even for characters where 1 byte is sufficient, different Unicode Codings were developed (UF-8, UTF-16 ve UTF-32)

String type in Python

- The string are defined as Unicode in Python without any coding.
- The string types can be defined with a single or double quote:

```
>>> a = 'EEE105'
>>> a = "EEE105"
```

- If the character sequence to be assigned to a string variable already contains a single/double quote, a backslash (\) should be used before it. If the string variable is defined with a single quote, the quote inside could de double or vice versa.

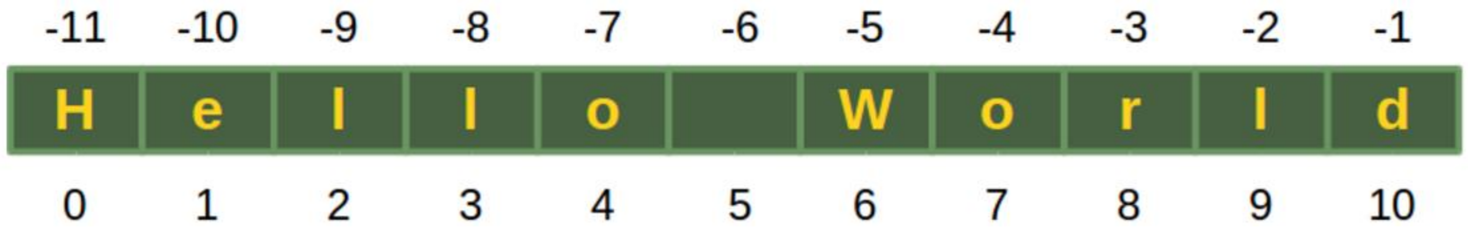
```
>>> a = 'EEE105\'s content'
>>> a = "EEE105\"s content"
```

- There is also a triple quote in Python which is used to define a multiline comment.

String type in Python

- A single character of a string variable in Python can be directly accesed with indexing.

```
>>> s = 'Hello World'
>>> s[0]
>>> 'H'
```



- The last characters can be accessed by using either of the following methods:

```
>>> s[len(s)-1]
>>> 'd'
>>> s[-1]
>>> 'd'
```

String type in Python

Concatenation:

- String concatenation is done by using operator "+":

```
>>> a = 'EEE105'
>>> b = " Computer Programming I"
>>> a+b
>>> 'EEE105 Computer Programming'
```

Repetition:

- A repetition of string is done using operator "*":

```
>>> a = 'AB'
>>> 3*a
>>> 'ABABAB'
```

String type in Python

Indexing:

- Indexing in Python is done through operator "[]".
- Python allows for negative indexing.

```
>>> a = 'AB'
>>> a[1]
>>> 'B'
>>> a[0]
>>> 'A'
>>> a[-1]
>>> 'B'
>>> a[-6]
>>> 'A'
>>> a[-7]
>>> Hata mesajı
```

**Indexing starts from 0!*

**After reaching the start of the variable it does not go back!*

String type in Python

Slicing:

- Slicing in Python is done through operators "[:]"
- The start/end indices take place on the left and right side of ":"

```
>>> a = 'Ankara'
>>> a[3:5]
>>> 'ar'
```

- The start/end indices can be left blank. In this case, it means from the start/to the end:

```
>>> a[:4]
>>> 'Anka'
>>> a[4:]
>>> 'ra'
```

**Dilimlemelerin indisleme gibi 0'dan başladığına ve de ikinci dilim indisinin dilime dahil olmadığına dikkat ediniz.*

String type in Python

Size & Length:

- The find the length of a string, len() function is used.
- "len" function gives the number of characters.
- "space" counts.
- To access the last character in a string variable a, the indexing a[len(a)-1] can be used.

```
 >>> a = 'Ankara'
>>> len(a)
>>> 6
 >>> a = 'Ankara İstanbul'
>>> len(a)
>>> 15
```

Mutable and Immutable Variables

- Mutable and Immutable variables are closely related to the concepts of "call by value" and "call by reference" which are also examined in the chapter about functions.
- In short, the string data type in Python is an immutable type. This means that the letters of a string cannot be modified by usual assignment.

```
>>> a = 'Ankara'
>>> a[0] = 'O'
error message .....
```

**it tries to change the string to "Onkara"*

How is a string variable kept in the memory?

- Almost anything in Python is an object and is kept in at a specific memory address. The content (value) of variables can be compared with the operator "==". But to check whether they are point at the same memory address, "is" operator is used:

```
>>> a = 'Ankara'; b = "Ankara"
>>> a == b
>>> True
>>> a is b
>>> True
>>> a = 'Med-Cezir'
>>> b = "Med-Cezir"
>>> a == b
>>> True
>>> a is b
>>> False
```

**They are pointing at the same object (the same memory address). Their contents are the same*

**They are not pointing at the same object (the same memory address). Their contents are the same*

String Variables in Python

Escape Sequences:

- String variables can contain special characters.
- They must have operator "\" to discriminate them against the usual characters.

Escape Sequence	Meaning Notes
\newline	Ignored
\\\\	Backslash (\\)
\\'	Single quote (')
\\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\N{name}	Character named name in the Unicode database (Unicode only)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\uxxxx	Character with 16-bit hex value xxxx (Unicode only)
\Uxxxxxxxx	Character with 32-bit hex value xxxxxxxx (Unicode only)
\v	ASCII Vertical Tab (VT)
\ooo	Character with octal value ooo
\xhh	Character with hex value hh

Variable Assignment

- The assignment operator is "=" as is in many programming languages.
- Python is a dynamicaly typed language. The content of the variable (its value) determines the data type.
- The very same variable can have different data types within the same code block.
- On the other hand, Python is a strongly typed language. Once the type is determined depending on the content, the operators should be compatible.

```
>>> a = "Gölbaşı"
>>> a = 27e12
>>> a = 1451 * 2321
```

Variable Assignment

- When we take into account that all the variables in Python are actually objects, caution should be exercised while assigning variables to one another.
- When we assign a value to a variable, a chunk of memory is allocated and an address of memory is assigned.
- When we assign variables to each other, only the memory address is assigned not their values.
- Unless delibaretly done, such phenomenon could have disastrous results. When the content of the assigned variable is modified, it also effects the first variable content.
- Python handles such a situation by assigning a new address during each value assignment.

```
>>> a = [2,4,5]
>>> b = a
>>> b[0] = 1
>>> a
>>> [1,4,5]
```

References

1 Wentworth, P., Elkner, J., Downey, A.B., Meyers, C. (2014). How to Think Like a Computer Scientist: Learning with Python (3nd edition).
2 Pilgrim, M. (2014). Dive into Python 3 by. Free online version: DiveIntoPython3.org ISBN: 978-1430224150.
3 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
4 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
5 Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/.
6 Millman, K.J., Aivazis, M. (2011). Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12.
7 John D. Hunter (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95.
8 Travis E. Oliphant (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20.
9 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
10 http://www.diveintopython.net/
11 https://docs.python.org/3/tutorial/
12 http://www.python-course.eu
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/