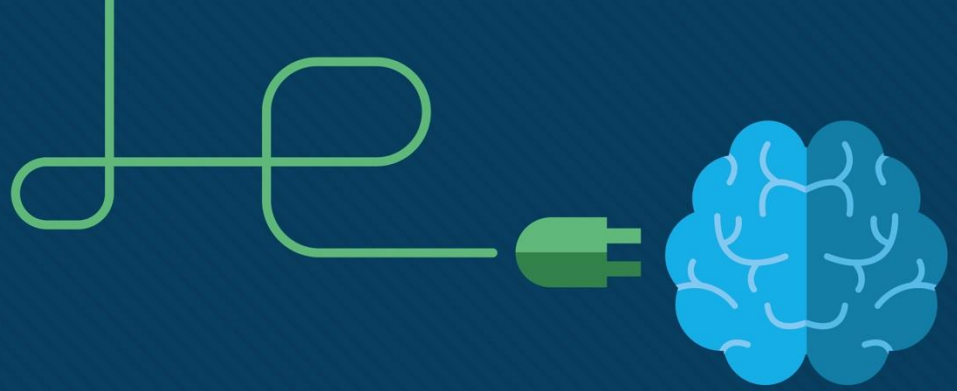


Programming Essentials in Python

Programming Essentials in Python

Module 1: Introduction to Python and Computer Programming

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Module Objectives

Module Title: Introduction to Python and computer programming

Objectives
The fundamentals of computer programming
Setting up your programming environment
Compilation vs. interpretation
Introduction to Python

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Programming – absolute basics

- **Instruction list** – a complete set of known commands, sometimes called **IL**

Four Elements of Language

- **Alphabet** - a set of symbols used to build words of a certain language (e.g., the Latin alphabet for English, the Cyrillic alphabet for Russian, Kanji for Japanese, and so on)
- **Lexis** - (aka a dictionary) a set of words the language offers its users (e.g., the word "computer" comes from the English language dictionary, while "cmoptrue" doesn't; the word "chat" is present both in English and French dictionaries, but their meanings are different)
- **Syntax** - a set of rules (formal or informal, written or felt intuitively) used to determine if a certain string of words forms a valid sentence (e.g., "I am a python" is a syntactically correct phrase, while "I a python am" isn't)
- **Semantics** - a set of rules determining if a certain phrase makes sense (e.g., "I ate a doughnut" makes sense, but "A doughnut ate me" doesn't)

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Programming – absolute basics

- **alphabetically** - a program needs to be written in a recognizable script, such as Roman, Cyrillic, etc.
- **lexically** - each programming language has its dictionary and you need to master it;
- **syntactically** - each language has its rules and they must be obeyed;
- **semantically** - the program has to make sense.

The Instruction list (IL) is, in fact, the alphabet of a machine language. This is the simplest and most primary set of symbols we can use to give commands to a computer. It's the computer's mother tongue.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Programming – absolute basics

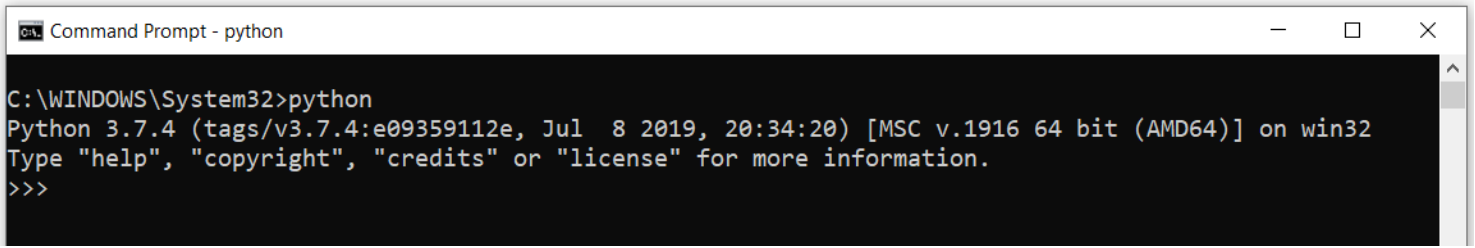
Compilation vs. Interpretation

	COMPILATION	INTERPRETATION
ADVANTAGES	• the execution of the translated code is usually faster; • only the user has to have the compiler - the end-user may use the code without it; • the translated code is stored using machine language - as it is very hard to understand it, your own inventions and programming tricks are likely to remain your secret.	• you can run the code as soon as you complete it - there are no additional phases of translation; • the code is stored using programming language, not the machine one - this means that it can be run on computers using different machine languages; you don't compile your code separately for each different architecture.
DISADVANTAGES	• the compilation itself may be a very time-consuming process - you may not be able to run your code immediately after any amendment; • you have to have as many compilers as hardware platforms you want your code to be run on.	• don't expect that interpretation will ramp your code to high speed - your code will share the computer's power with the interpreter, so it can't be really fast; • both you and the end user have to have the interpreter to run your code.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Downloading and installing Python

- Download and install Python3 https://www.python.org/downloads/
- Starting Python in Windows



- Windows PATH error ('python' is not recognized as an internal or external command)

https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/

Installing Python3 using Anaconda Package Manager

https://www.anaconda.com/distribution/#download-section

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Module 2: Data types, variables, basic input-output operations, basic operators

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Module Objectives

Module Title: Data types, variables, basic input-output operations, basic operators

Objectives
data types and the basic methods of formatting, converting, inputting and outputting data;
operators;
variables.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Built-in functions()

print(“Hello, NSD Students”)

function_name(argument(s))

- A function is a block of organized reusable code that performs an action.
- A function has a name and is called, or executed, by that name.
- Optionally, functions can accept arguments and return data.

A function may have:

- an **effect**
- a **result**

Built-in Functions¶

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Reference: https://docs.python.org/3/library/functions.html

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

functions()

print(“Hello, NSD Students”)

- First, Python checks if the name specified is **legal** (it browses its internal data in order to find an existing function of the name; if this search fails, Python aborts the code);
- second, Python checks if the function's requirements for the number of arguments **allows you to invoke** the function in this way (e.g., if a specific function demands exactly two arguments, any invocation delivering only one argument will be considered erroneous, and will abort the code's execution);
- third, Python **leaves your code for a moment** and jumps into the function you want to invoke; of course, it takes your argument(s) too and passes it/them to the function;
- fourth, the function **executes its code**, causes the desired effect (if any), evaluates the desired result(s) (if any) and finishes its task;
- finally, Python **returns to your code** (to the place just after the invocation) and resumes its execution.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The print() function - instructions

```
print("The itsy bitsy spider climbed up the waterspout.")
print()
print("Down came the rain and washed the spider out.")
```

Python's syntax is quite specific in this area. Unlike most programming languages, Python requires that **there cannot be more than one instruction in a line**.

A line can be empty (i.e., it may contain no instruction at all) but it must not contain two, three or more instructions. This is strictly prohibited.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The print() function - instructions

The print() function - the escape and newline characters - \n

```
print("The itsy bitsy spider\nclimbed up the waterspout.")
```

The print() function - using multiple arguments

```
print("The itsy bitsy spider" , "climbed up" , "the waterspout.")
```

The print() function - the positional way of passing the arguments

```
print("My name is", "Python", "Monty")
```

The print() function - the keyword arguments

```
print("My name is", "Python.", end=" ")
print("Monty Python.")

print("My", "name", "is", "Monty", "Python.", sep="-")
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Key takeaways

Key takeaways

1. The `print()` function is a **built-in** function. It prints/outputs a specified message to the screen/consol window.

2. Built-in functions, contrary to user-defined functions, are always available and don't have to be imported. Python 3.7.1 comes with 69 built-in functions. You can find their full list provided in alphabetical order in the Python Standard Library.

3. To call a function (**function invocation**), you need to use the function name followed by parentheses. You can pass arguments into a function by placing them inside the parentheses. You must separate arguments with a comma, e.g., `print("Hello,", "world!")`. An "empty" `print()` function outputs an empty line to the screen.

4. Python strings are delimited with **quotes**, e.g., `"I am a string"`, or `'I am a string, too'`.

5. Computer programs are collections of **instructions**. An instruction is a command to perform a specific task when executed, e.g., to print a certain message to the screen.

6. In Python strings the **backslash** (`\`) is a special character which announces that the next character has a different meaning, e.g., `\n` (the **newline character**) starts a new output line.

7. **Positional arguments** are the ones whose meaning is dictated by their position, e.g., the second argument is outputted after the first, the third is outputted after the second, etc.

8. **Keyword arguments** are the ones whose meaning is not dictated by their location, but by a special word (keyword) used to identify them.

9. The `end` and `sep` parameters can be used for formatting the output of the `print()` function. The `sep` parameter specifies the separator between the outputted arguments (e.g., `print("H", "E", "L", "L", "O", sep="-")`, whereas the `end` parameter specifies what to print at the end of the print statement.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The other functions

The len() function

```
1  #The len() function
2
3  city = "Winnipeg"
4  city_len = len(city)
5  print(city_len)
```

```
8
```

Nesting functions

```
1  #Nesting functions
2
3  city = "Winnipeg"
4  print(len(city))
5
6  print(len("Toronto"))
```

```
8
7
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The other functions

The Functions that run against an object - Methods

String Methods

- Everything in Python is an object.
- Every object has a type.
- “Winnipeg” is an object of type "str".
- “Winnipeg” is a string object.
- city = “Winnipeg”.
  city is a string object.

Methods are functions run against an object.

`object.method()`

```
1  #String Methods
2
3  city = "Winnipeg"
4  print(city.lower())
5  print(city.upper())
6
7  print("toronto".capitalize())
```

```
winnipeg
WINNIPEG
Toronto
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The other functions

String Concatenation

```
1  #String Concatenation
2
3  print("I " + "love " + "Winnipeg")
4
5  one = "I"
6  two = "love"
7  three = "Winnipeg"
8  print(one + " " + two + " " + three)
9
10 print("{} {} {}" . format(one, two, three))
```

```
I love Winnipeg
I love Winnipeg
I love Winnipeg
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The other functions

Formatting Strings

```
1  # Formatting Strings
2
3  print("I {} Winnipeg." . format("love"))
4  print("{} {} {}." . format("I", "love", "Winnipeg"))
5
6  print("I {0} {1}. {1} {0}s me." . format("love", "Winnipeg"))
```

```
I love Winnipeg.
I love Winnipeg.
I love Winnipeg. Winnipeg loves me.
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The other functions

Repeating Strings

```
1  # Repeating Strings
2
3  print("*" * 8)
4  print("Love " * 5)
```

```
********
Love Love Love Love Love
```

The str() Function

```
1  # The str() Function
2
3  version = 7
4  print("I love CCNA " + str(version) + ".")
5
```

```
I love CCNA 7.
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Data types

Literals – the data in itself; data whose values are determined by the literal itself.

Take a look at the following set of digits:

```
123
```

Can you guess what value it represents? Of course you can - it's *one hundred twenty three*.

But what about this:

```
c
```

Does it represent any value? Maybe. It can be the symbol of the speed of light, for example. It also can be the constant of integration. Or even the length of a hypotenuse in the sense of a Pythagorean theorem. There are many possibilities.

Example:

```
print(123)
print(c)
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Data types

Integers

- integers (octal and hex)
- floats

```
print(0o73)                          59
print(0x123)                         291
print(hex(291))                      0x123
print(10 / 3)                        3.3333333333333335
print(float(10 / 3))                 3.3333333333333335
print(int(10 / 3))                   3
print(round(float(10 / 3), 2))       3.33
print(round((10 / 3), 2))            3.33
print(0xAF / 0X12)                   9.722222222222221
print(round(0xAF / 0X12))            10
print(10 / 2.0)                      5.0
print(int(10 / 2.0))                 5
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Data types

Scientific Notation

3 x 10^8 is equivalent to 3E8 in Python

6.95 x 10^-34 is equivalent to 6.95E-34 in Python

Note:

the exponent (the value after the E) has to be an integer;

the base (the value in front of the E) may be an integer.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Data types

Strings

- are used when you need to process text.
- should be enclosed by either quote or apostrophe

Examples:

```
print("I am learning Python language.")
print('Is it fun to learn Python?')
print("Maybe, we'll find out.")
print('')
print("Cool!, \"Python is powerfull\".")
print("")
print('Cool!, "This is another way of doing it".')
```

```
I am learning Python language.
Is it fun to learn Python?
Maybe, we'll find out.

Cool!, "Python is powerfull".

Cool!, "This is another way of doing it".
```

Boolean values

- used to represent an abstract value – truthfulness
- use two distinct values; True and False
- denoted as 1 and 0

```
print(1 < 2)
print(3 < 2)
print(1 == 1)
```

```
True
False
True
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Data types

Indexing

String:	W	i	N	n	l	p	E	g
Index:	0	1	2	3	4	5	6	7
Index:	-8	-7	-6	-5	-4	-3	-2	-1

```
city = "WiNnIpEg"
print(city[0]) # index 0
print(city[0:8]) # index from 0 to 8, exclude last
print(city[:3]) # index from start to 3, exclude last
print(city[5:]) # index from 5 to last
print(city[-3:]) # index from -3 to last
print(city[:-5]) # index from start to -5, exclude last
```

```
C:\Users\rogelio.villaver\
W
WiNnIpEg
WiN
pEg
pEg
WiN
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Operators

Arithmetic operators: exponentiation

A `**` (double asterisk) sign is an **exponentiation** (power) operator. Its left argument is the **base**, its right, the **exponent**.

Classical mathematics prefers notation with superscripts, just like this: 2^3. Pure text editors don't accept that, so Python uses `**` instead, e.g., `2 ** 3`.

Arithmetic operators: multiplication

An `*` (asterisk) sign is a **multiplication** operator.

Arithmetic operators: integer division

A `//` (double slash) sign is an **integer divisional** operator. It differs from the standard `/` operator in two details:

- its result lacks the fractional part - it's absent (for integers), or is always equal to zero (for floats); this means that **the results are always rounded**;
- it conforms to the *integer vs. float rule*.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Operators

Operators: remainder (modulo)

Its graphical representation in Python is the `%` (percent) sign,

```
print(14 % 4)
```

As you can see, the result is two. This is why:

- `14 // 4` gives `3` → this is the integer **quotient**;
- `3 * 4` gives `12` → as a result of **quotient and divisor multiplication**;
- `14 - 12` gives `2` → this is the **remainder**.

```
print(12 % 4.5)
```

Operators: addition

The **addition** operator is the `+` (plus) sign, which is fully in line with mathematical standards.

```
print(-4 + 4)
print(-4. + 8)
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Operators

The subtraction operator, unary and binary operators

The **subtraction** operator is obviously the `-` (minus) sign, although you should note that this operator also has another meaning - **it can change the sign of a number**.

```
print(-4 - 4)
print(4. - 8)
print(-1.1)
```

By the way: there is also a unary `+` operator. You can use it like this:

```
print(+2)
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Operators

Operators and their priorities

Consider the following expression:

```
2 + 3 * 5
```

List of priorities

Priority	Operator	
1	`+`, `-`	unary
2	`**`	
3	`*`, `/`, `%`	
4	`+`, `-`	binary

Note: we've enumerated the operators in order **from the highest (1) to the lowest (4) priorities**.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Variables

If you want to **give a name to a variable**, you must follow some strict rules:

- the name of the variable must be composed of upper-case or lower-case letters, digits, and the character _ (underscore)
- the name of the variable must begin with a letter;
- the underscore character is a letter;
- upper- and lower-case letters are treated as different (a little differently than in the real world - *Alice* and *ALICE* are the same first names, but in Python they are two different variable names, and consequently, two different variables);
- the name of the variable must not be any of Python's reserved words (the keywords - we'll explain more about this soon).

- Case sensitive. (Case matters!)
  Alice and ALICE are different variables.
- Must start with a letter.
- Can contain numbers.
- Underscores allowed in variable names
- Not allowed: “+” “-” signs

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Keywords

All the keywords except `True`, `False` and `None` are in lowercase and they must be written as they are. The list of all the keywords is given below.

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

They are called **keywords** or (more precisely) **reserved keywords**. They are reserved because **you mustn't use them as names**: neither for your variables, nor functions, nor any other named entities you want to create.

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Variables

Creating variables

What can you put inside a variable?

Anything.

You can use a variable to store any value of any of the already presented kinds, and many more of the ones we haven't shown you yet

The value of a variable is what you have put into it. It can vary as often as you need or want. It can be an integer one moment, and a float a moment later, eventually becoming a string.

Let's talk now about two important things - **how variables are created**, and **how to put values inside them** (or rather - how to give or **pass values** to them).

REMEMBER

A variable comes into existence as a result of assigning a value to it. Unlike in other languages, you don't need to declare it in any special way.

If you assign any value to a nonexistent variable, the variable will be **automatically created**. You don't need to do anything else.

The creation (or otherwise - its syntax) is extremely simple: **just use the name of the desired variable, then the equal sign (=) and the value you want to put into the variable.**

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Variables

Examples

```
a = 1
b = 2
print(a)
print(b)
c = "NSD is awesome"
print(c)
b = 3
print(b)
d = b
print(d)
b = 4
print(d)
e = c
print(e)
c = "NSD is cool"
print(e)
```

```
1
2
NSD is awesome
3
3
3
NSD is awesome
NSD is awesome

Process finished with exit code 0
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Variables

Assigning a new value to an already existing variable

```
var = 1
print(var)
var = var + 1
print(var)
var += 1
print(var)
```

```
1
2
3

Process finished with exit code 0
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Variables

Shortcut operators

If `op` is a two-argument operator (this is a very important condition) and the operator is used in the following context:

```
variable = variable op expression
```

It can be simplified and shown as follows:

```
variable op= expression
```

Take a look at the examples below. Make sure you understand them all.

`i = i + 2 * j` ⇒ `i += 2 * j`

`var = var / 2` ⇒ `var /= 2`

`rem = rem % 10` ⇒ `rem %= 10`

`j = j - (i + var + rem)` ⇒ `j -= (i + var + rem)`

`x = x ** 2` ⇒ `x **= 2`

Basic input-output

The `input()` function

→ `input()`

```
print("Enter your first name: ")
first = input()
print("Enter your last name: ")
last = input()
print("Welcome!", first, last)
print(f"Welcome! {first} {last}")
print("Welcome! {} {}" .format(first, last))
print("Welcome! " + first + " " + last + ".")
```

```
Enter your first name:
Rogelio
Enter your last name:
Villaver
Welcome! Rogelio Villaver
Welcome! Rogelio Villaver
Welcome! Rogelio Villaver
Welcome! Rogelio Villaver.
```

https://docs.python.org/3/tutorial/inputoutput.html

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Basic input-output

The `input()` function with an argument and type casting

```
input_num = int(input("Enter a number: "))
number = input_num ** 2.0
print(input_num, "to the power of 2 is", number)
```

```
Enter a number: 10
10 to the power of 2 is 100.0
```

```
input_num = int(input("Enter a number: "))
number = input_num ** 2.0
print(input_num, "to the power of 2 is", number)
```

```
Enter a number: Rogelio

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-b3f2733dd7bb> in <module>
----> 1 input_num = int(input("Enter a number: "))
      2 number = input_num ** 2.0
      3 print(input_num, "to the power of 2 is", number)

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

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Basic operators

String operators - introduction

Concatenation

The `+` (plus) sign, when applied to two strings, becomes a **concatenation operator**:

```
string + string
```

Replication

The `*` (asterisk) sign, when applied to a string and number (or a number and string, as it remains commutative in this position) becomes a **replication operator**:

```
string * number
number * string
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Basic operators

String operators - introduction

Type conversion: `str()`

You already know how to use the `int()` and `float()` functions to convert a string into a number.

This type of conversion is not a one-way street. You can also **convert a number into a string**, which is way easier and safer - this operation is always possible.

A function capable of doing that is called `str()`:

```
str(number)
```

```
a = float(input("Input a side length: "))
b = float(input("Input b side length: "))
print("Hypotenuse length is " + str((a**2 + b**2) ** .5))
```

```
Input a side length: 5
Input b side length: 6
Hypotenuse length is 7.810249675906654
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential


CISCO