Python: programming language

Asst. Prof. Dr. Yalçın İŞLER

Textbooks

- “Python 3” from tutorialspoint.com, 2016.
- “Çocuklar için Uygulamalarla Python” from Ahmet Aksoy (abaküs), 2018.

History

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python 1.0 (November 1994), Python 2.0 (2000), Python 3 (2008).
- Latest versions: Python 2.7.16, Python 3.7.3.
- Python 3 has no backward compatibity to Python 2.

Some differences between Phython 2 and Python 3

- Printing with use of parantheses

```
print "Hello World"      #is acceptable in Python 2
print("Hello World")     # in Python 3, print must be followed by ()
```

- Printing with no line-feeding

```
print x,      # Trailing comma suppresses newline in Python 2
print(x,end=" ")     # Appends a space instead of a newline in Python 3
```

Features

- Easy-to-learn
- Easy-to-read
- Easy-to-maintain
- A broad standard library
- Interactive Mode
- Portable
- Extendable
- Databases
- GUI Programming
- Scalable

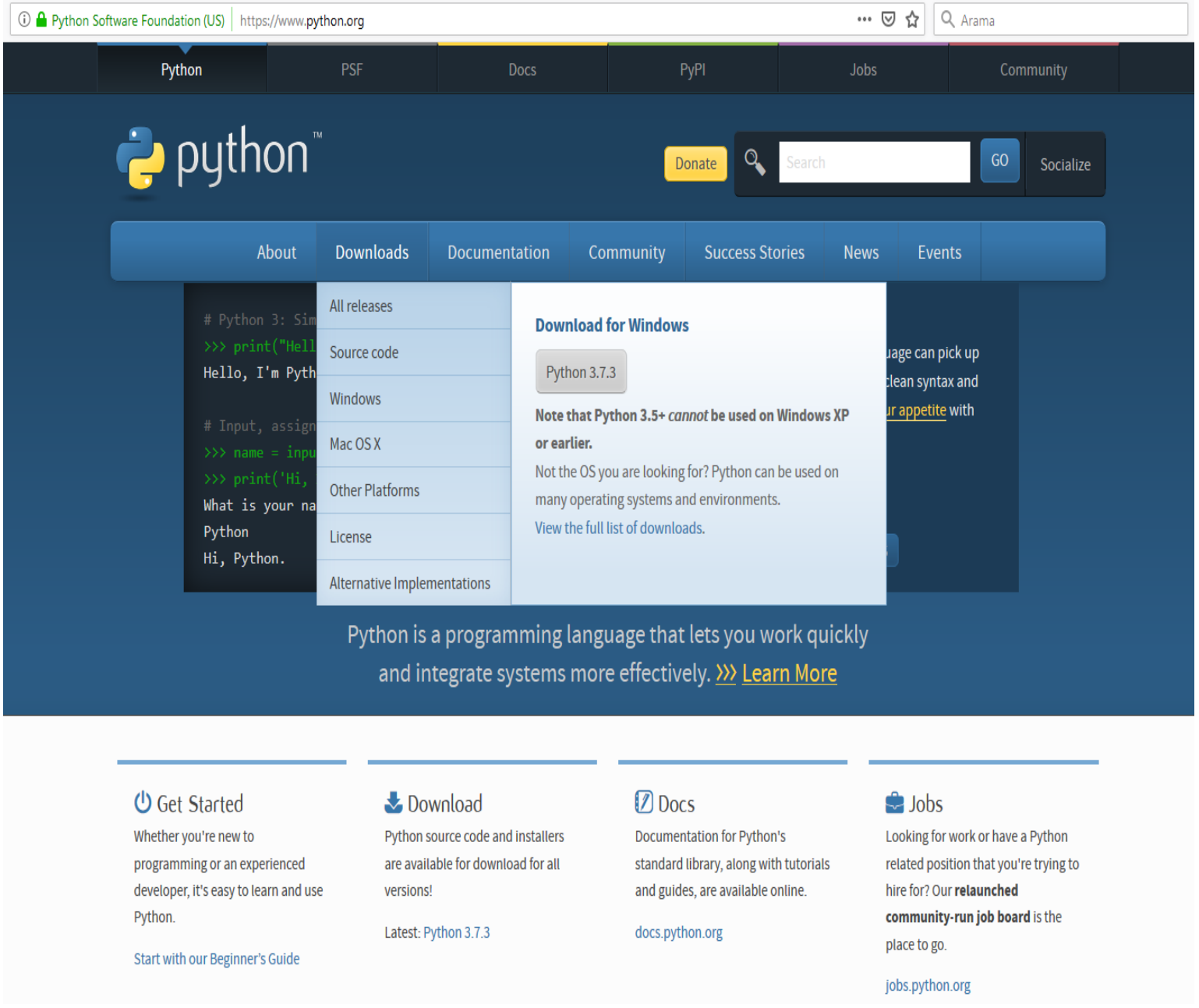
Why we should prefer

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

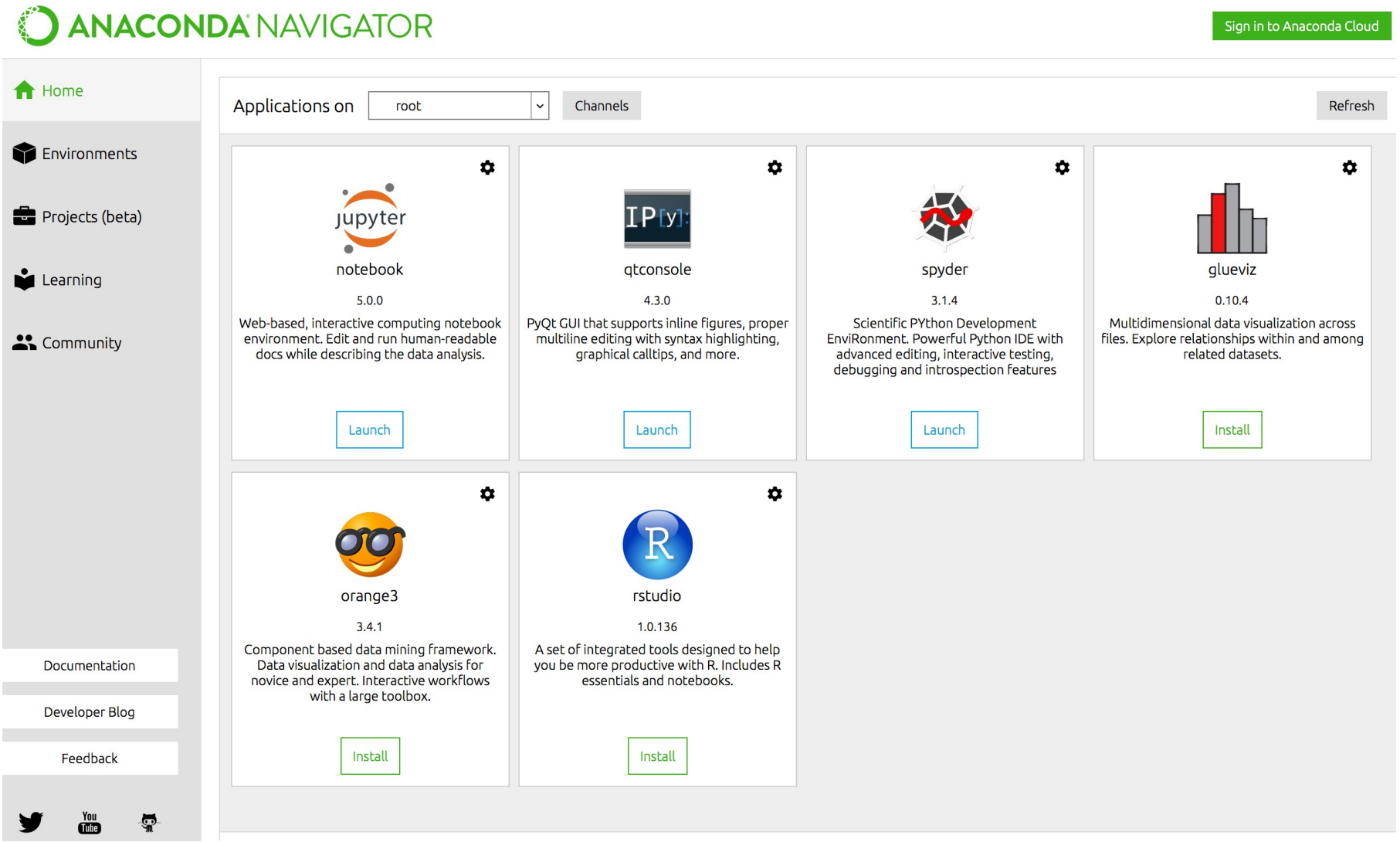
Applications

- Basic arithmetic calculations
- Complex calculations, problem solving
- Scientific computing, Statistics
- Artificial intelligence, Machine learning, Deep learning
- Desktop applications
- Web applications
- Mobile applications
- Database applications
- Game development
- Electronic board programming (Raspberry PI, Arduino, ...)

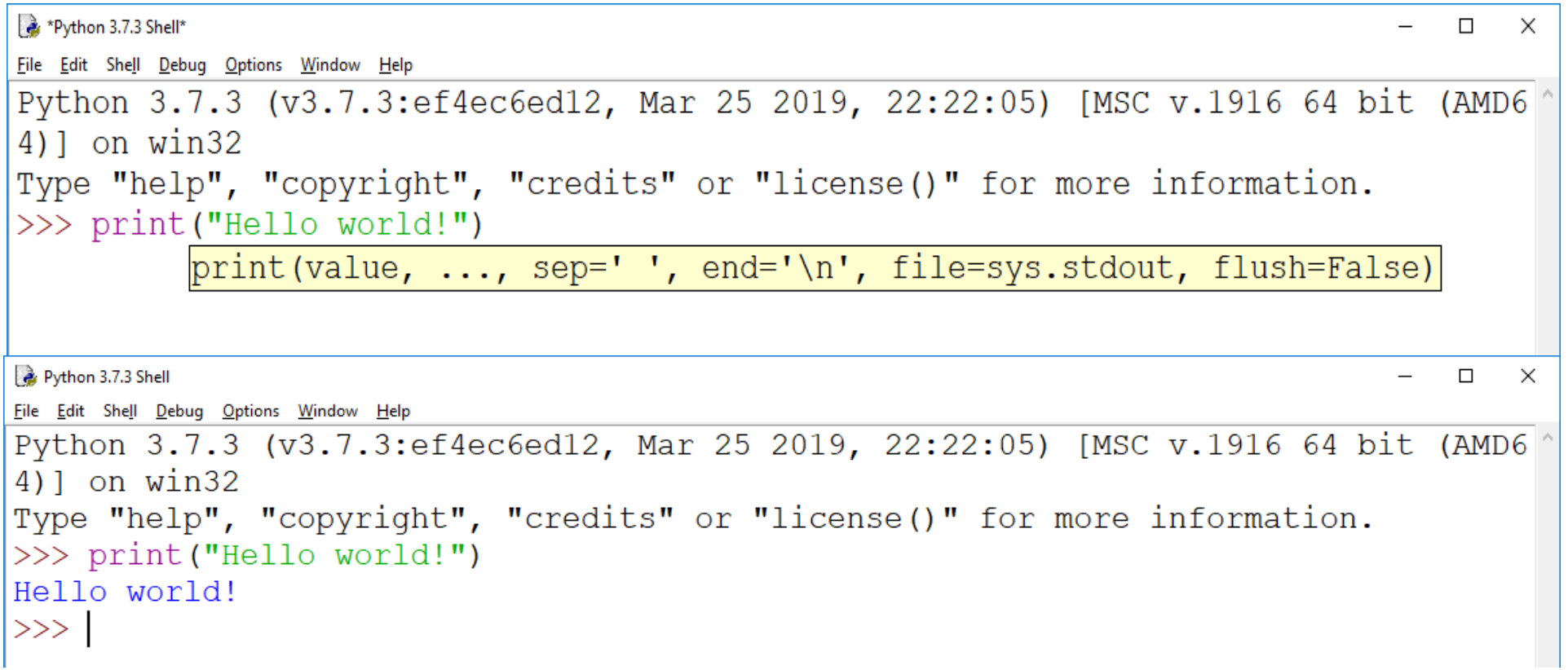
Download: www.python.org



Anaconda

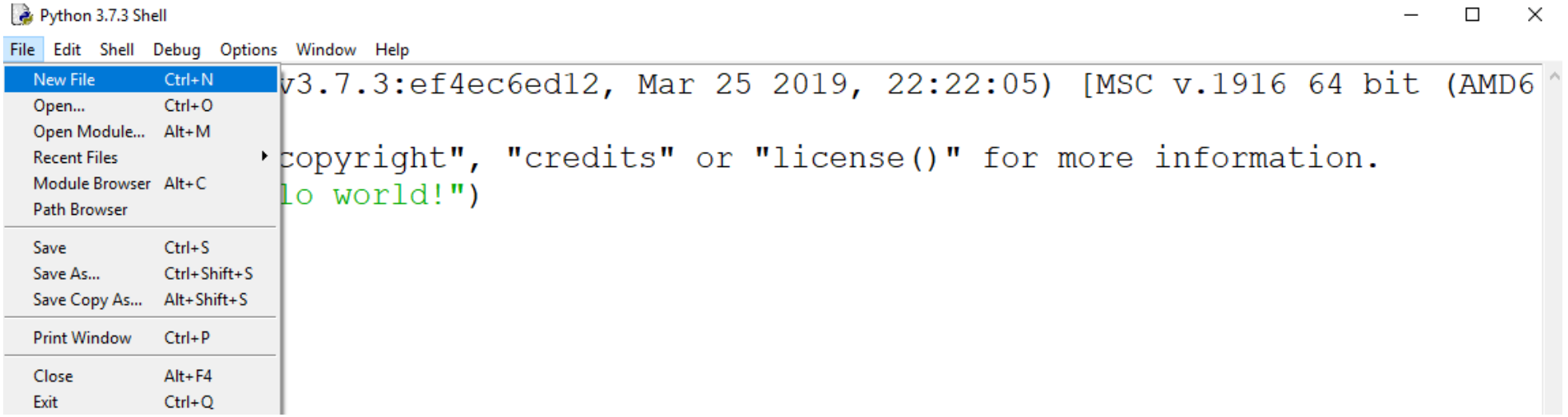

ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Projects (beta)
Learning
Community
Documentation
Developer Blog
Feedback
Applications on
root
Channels
Refresh
notebook
5.0.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
qtconsole
4.3.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
spyder
3.1.4
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch
glueviz
0.10.4
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install
orange3
3.4.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install
rstudio
1.0.136
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.
Install

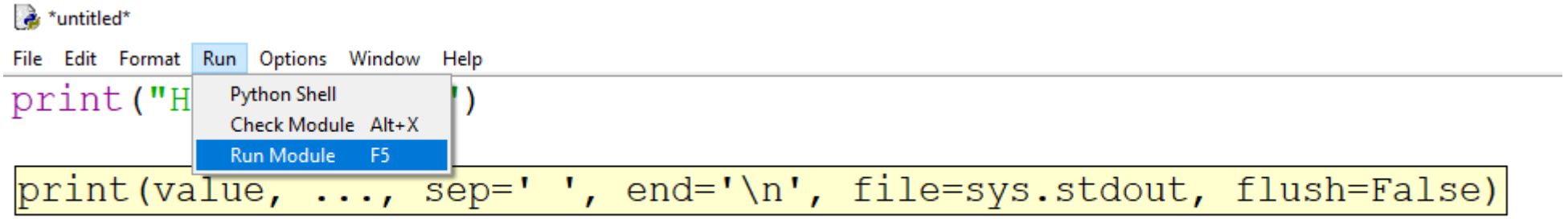
Interactive mode programming: python



If you are running the older version of Python (Python 2.x), use of parenthesis as in print function is optional.

Script mode programming: IDLE





- python say_hello.py

Linux trick:

- Assuming that you have Python interpreter available in the /usr/bin directory. Now, try to run this program as follows:
- # This is to make file executable:

$ chmod +x say_hello.py

$ ./say_hello.py

- This produces the following result:

Hello, Python!

Identifiers

- An identifier is a name used to identify a variable, function, class, module or other Python object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language. Thus, x and X are different in Python.

Naming convention

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strong private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Identation & blocks

- No braces “ { and } “ to indicate blocks of code and definitions or flow control.
- Similarly no Begin and End.
- The number of spaces in the indentation is variable, but all statements within the same block must be indented the same amount.

Identation examples

```
if True:
    print ("True")
else:
    print ("False")
```

```
if True:
    print ("True")
else:
  print ("False")
```

```
if True:
    print ("Answer")
    print ("True")
else:
    print "(Answer")
  print ("False")
```

Multi-line statements

- Statements in Python typically end with a new line. Python, however, allows the use of the line continuation character (\) to denote that the line should continue.
- The statements contained within the [ ], { }, or () brackets do not need to use the line continuation character.

- total = item_1 + \
  item_2 + \ item_3
- days = ['Monday',
  'Tuesday',
  'Wednesday',
  'Thursday', 'Friday']

Quotations

- Python accepts single ('), double (") and triple ''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines.

- word = 'word'
- sentence = "This is a sentence."
- paragraph = """This is a paragraph. It is made up of multiple lines and sentences."""

Commenting

- A hash sign (#) that is not inside a string literalis the beginning of a comment. All characters after the #,up to the end of the physical line,are part of the comment and the Python interpreter ignores them:

```
# First comment
print("Hello, Python!") # second comment
url ="www.islerya.com" # This is again comment
```

- Python does not have multiple-line commenting feature :(. You have to comment each line individually:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Blank lines

- A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.
- In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

User input, semicolon

- Displaying “Press the enter key to exit”, and then waits for the user to take action:

```
#!/usr/bin/python3

input("\n\nPress the enter key to exit.")
```

- Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

- The semicolon (;) allows multiple statements on a single line given that no statement starts a new code block:

```
import sys; x ='foo'

sys.stdout.write(x +'\n')
```

Assigning values

```
counter =100 # integer
miles   =1000.0 # float
name    ="Yalcin" # string
print(counter)
print(miles)
print(name)
del name
```

```
a =b =c =1
```

- Here, an integer object is created with the value 1, and all the three variables are assigned to the same memory location.

```
a,b,c =1,2,"Yalcin"
```

- Here, two integer objects with values 1 and 2 are assigned to the variables a and b respectively,and one string object with the value "Yalcin" is assigned to the variable c.

Numbers and strings

```
>>> Var1 = 10 #integer
>>> Var2 = 10.0 #float
>>> Var3 = 2+3j #complex
>>> print(Var1)
10
>>> print(Var2)
10.0
>>> print(Var3)
(2+3j)
```

```
>>> str ='Hello World!'
>>> print(str)
Hello World!
>>> print(str[0])
H
>>> print(str[2:5])
llo
>>> print(str[2:])
llo World!
>>> print(str*2)
Hello World!Hello World!
>>> print("Yalcin said '"+str+"'")
Yalcin said 'Hello World!'
>>>
```

Lists

```
list = [ 'abcd', 786 , 2.23, 'yalcin', 70.2 ]
tinylist = [123, 'isler']
print (list)     # Prints complete list
print (list[0]) # Prints first element of the list
print (list[1:3])  # Prints elements starting from 2nd till 3rd
print (list[2:])   # Prints elements starting from 3rd element
print (tinylist * 2)  # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

```
Hello world!
['abcd', 786, 2.23, 'yalcin', 70.2]
abcd
[786, 2.23]
[2.23, 'yalcin', 70.2]
[123, 'isler', 123, 'isler']
['abcd', 786, 2.23, 'yalcin', 70.2, 123, 'isler']
```

Dictionaries

```
>>> tinydict ={'name':'YALCIN','code':1234,'dept':'BIOMEDICAL'}
>>> print(tinydict)
{'name': 'YALCIN', 'code': 1234, 'dept': 'BIOMEDICAL'}
>>> print(tinydict.keys())
dict_keys(['name', 'code', 'dept'])
>>> print(tinydict.values())
dict_values(['YALCIN', 1234, 'BIOMEDICAL'])
>>> print(tinydict['name'])
YALCIN
```

- Dictionaries are enclosed by curly braces { } and values can be assigned and accessed using square braces [ ]

Arithmetic & Comparison operators

- addition + subtraction - multiplication * division / modulus % power ** floor division //

- equal == not equal != greater > less < greater-equal >= less-equal <=

Assignment & Bitwise & Logical operators

- = += -= *= /= %= **= //=

- And & Or | Ex-or ^
- Not (1’s complement) ~
- Shift-Left << Shift-Right >>

- X and Y X or Y not X