Basic Python for Quantum

Muhammad Imansyah Basudewa

The University Center of Excellence for Advanced Intelligent Communications (AICOMS),
School of Electrical Engineering, Telkom University, Bandung, INDONESIA
E-mail: {Your E-mail Address}

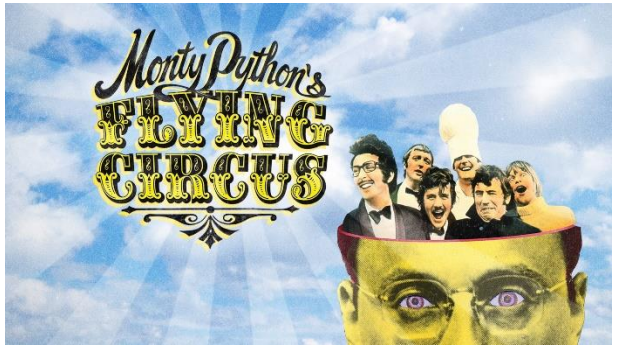
Presented at
The 2021 AICOMS Workshop on Quantum Technology: Theory, Development, and Business (AICOMS-Q)
Online, September 19, 2021

Outline

Introduction [1/5]

- Python is a high level language, multipurpose, object oriented programming, and interactive environment.
- Python is a scripting language that can be used for website development, ML, AI, data science, data visualization, business applications and many more.
- We use Python 3 (3.6+) as our programming language and we will be working on the library Qiskit!

Introduction [2/5]



python SOFTWARE FOUNDATION

python™

Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered

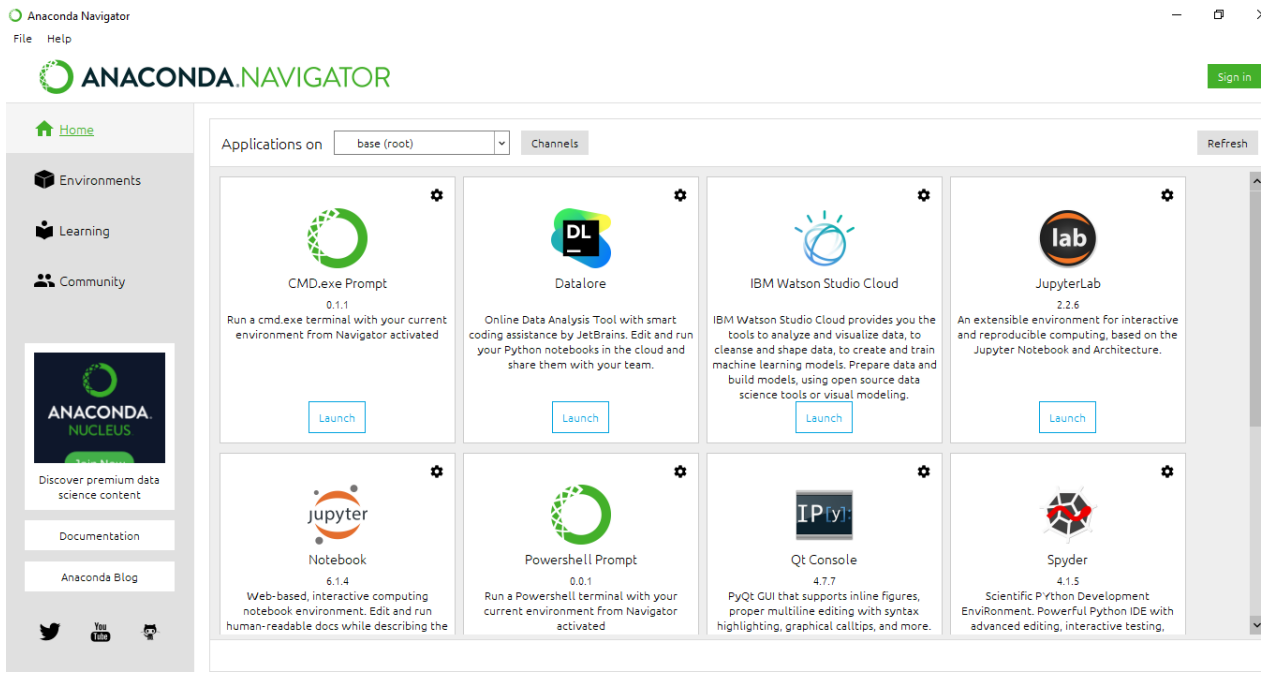
- Guido Van Rossum

Introduction [3/5]

- Jupyter notebook is one of the most popular tools to create and share documents that contain interactive code, visualizations, text, etc.
- Anaconda is a free and open source distribution of the Python languages that aims to simplify package management and deployment.

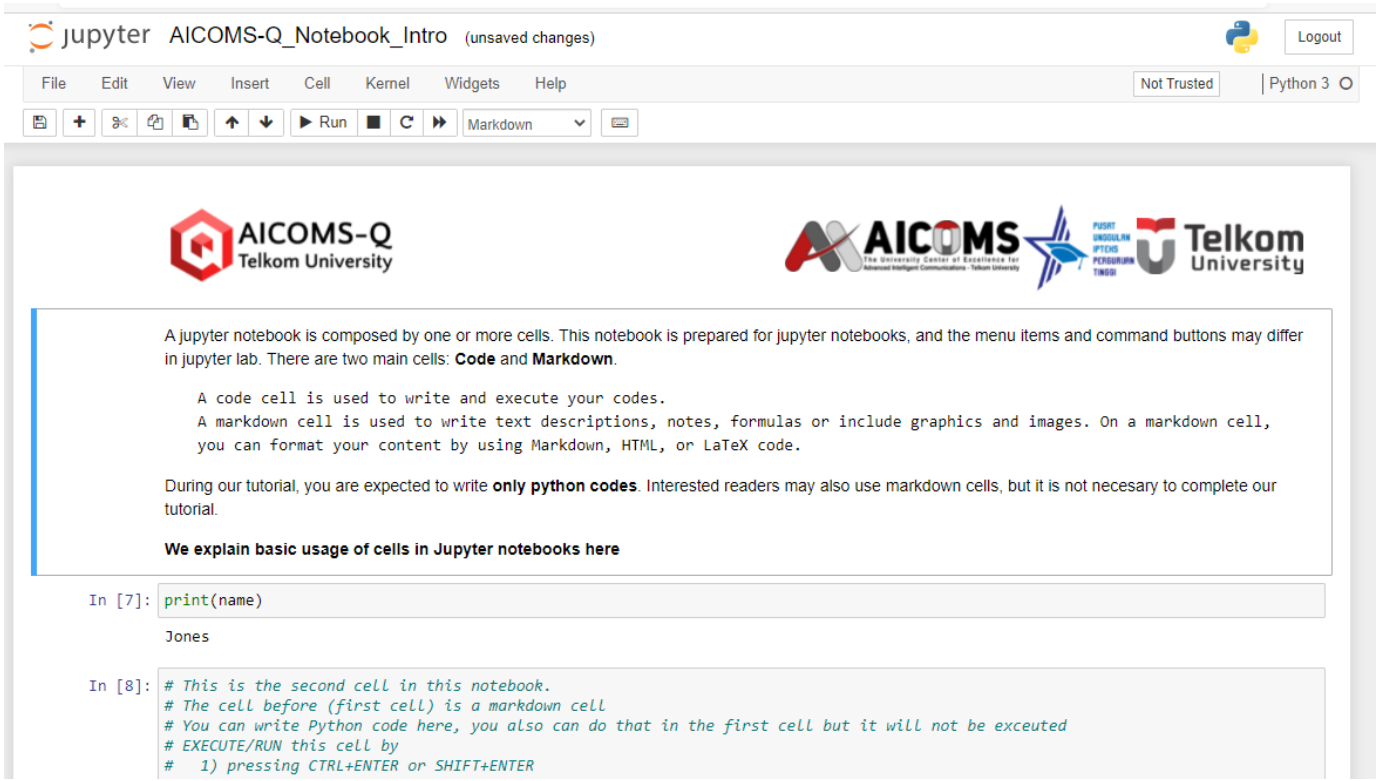
https://docs.anaconda.com/anaconda/navigator/getting-started/

Introduction [4/5]



- **Anaconda Navigator interface**
- Navigator is a desktop graphical user interface that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands.
- The following applications are available by default in navigator
- JupyterLab, Jupyter Notebook, Spyder, PyCharm, VSCode, Glueviz, Orange 3 App, R Studio, Anaconda Prompt (Windows only), Anaconda PowerShell (Windows Only)

Introduction [5/5]



- The notebook user interface
- **Notebook kernel**
  Computational engine that executes the code contained in a notebook.
- **Notebook cell**
  Container for text to be displayed in a notebook or for code to be executed by the notebook's kernel.
- **Code** to be executed in the kernel. In front of the code cells are brackets that indicate the order in which the code was executed.
- In [ ]: indicates that the code has not yet been executed.
- In [*]: indicates that the execution has not yet been completed

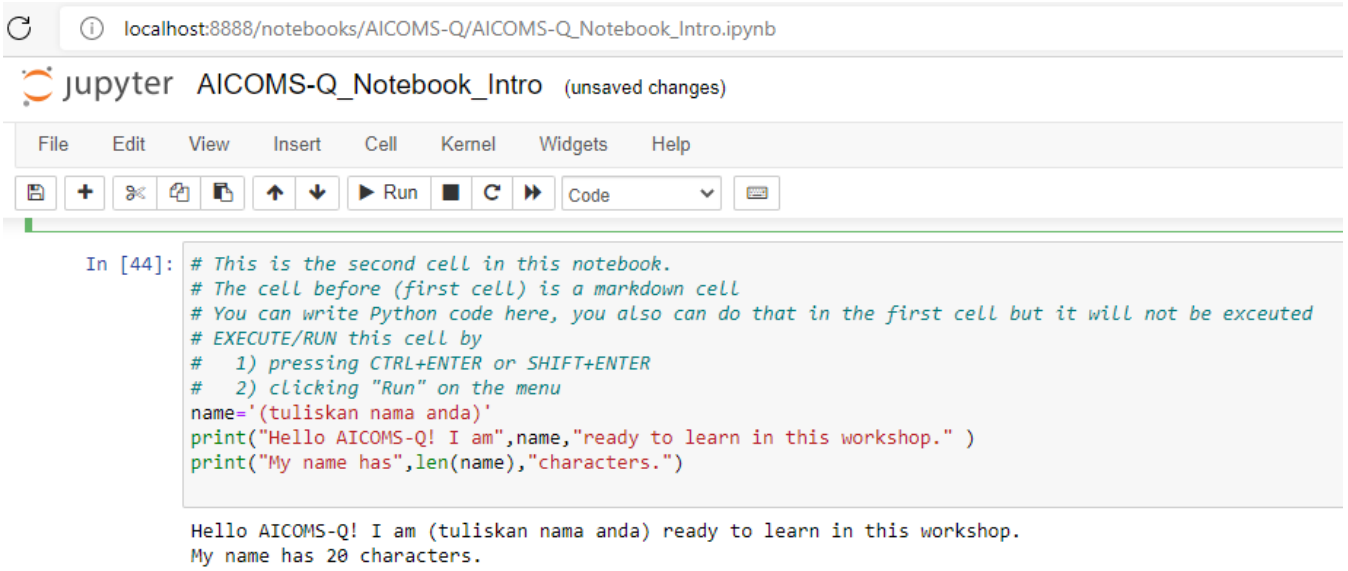
The Basics [1/7]

In [31]:
```
x = 34 - 23     #A comment
y = "Hello"
z = 3.45
if z ==3.45 or y == "Hello":
    x = x+1
    y = y + "World"    #String concat.
print(x)
print(y)
```

```
12
HelloWorld
```

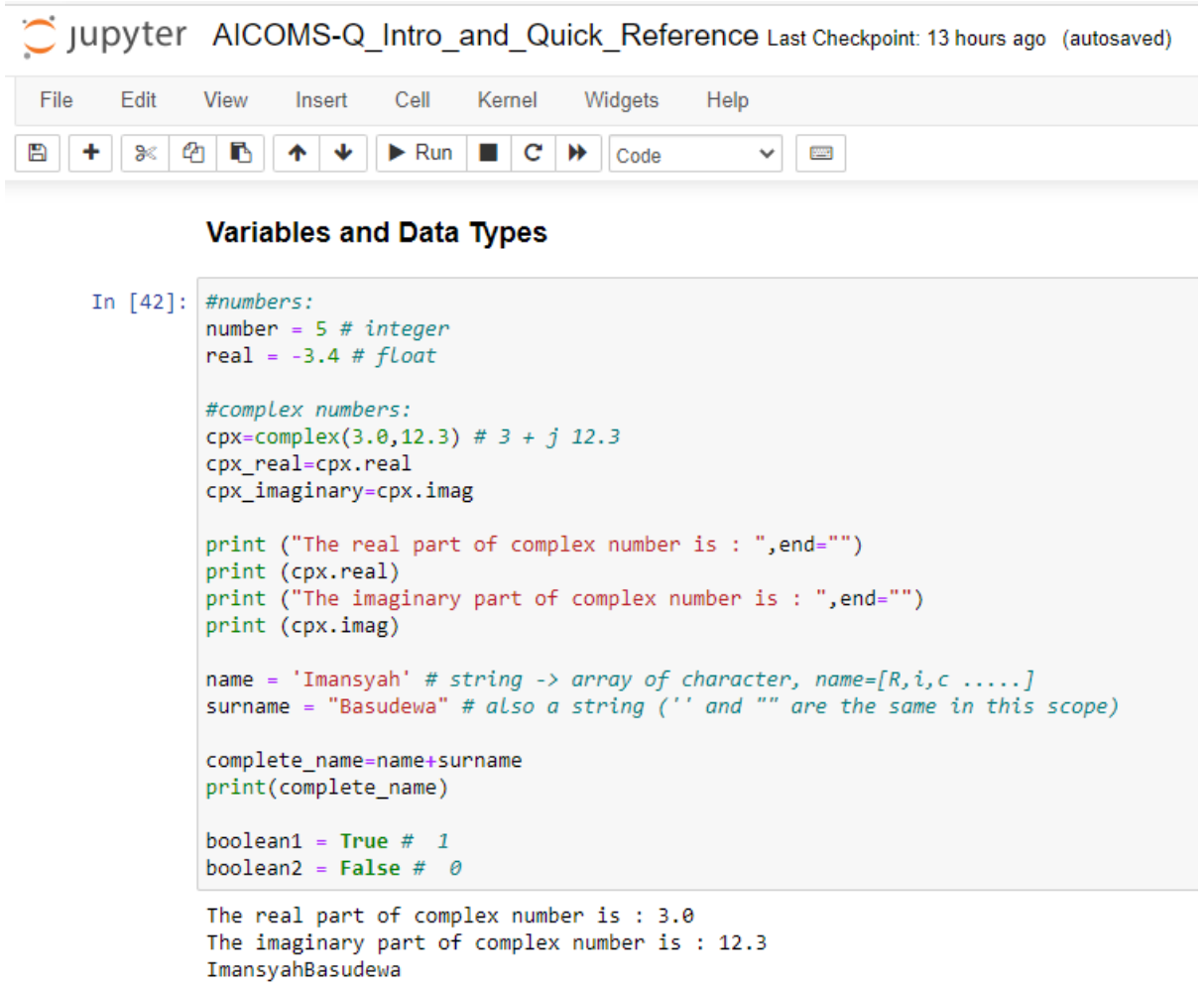
- Indentation matters to code meaning
- First assignment to a variable creates it
- Variable types don’t need to be declared
- Python figures out the variable types on its own
- Assignment is = and comparison is ==
- For number + - / * % are expected
- Logical operators are words (and, or, not) not symbols
- The basic printing command is “print”

The Basics [2/7]



- **Comments**
- Start comments with hash character (#) - the rest line is ignored
- Can include a “documentation string” as the first line of any new function or class that you define
- *#This is also a comment in python*
- “”” This is an example of multiline comment that spans multilines …”””

The Basics [3/7]



Variable and Basic datatypes

- Integer

  z = 5
- Float

  x = -3.4
- String
- Can use “” or ‘’ to specify.

  “abc” ‘abc’ (same thing).
- Unmatched can occur within the string.

  “matt’s”
- Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them

  “””a ‘b”c “””

The Basics [4/7]

Naming Rules

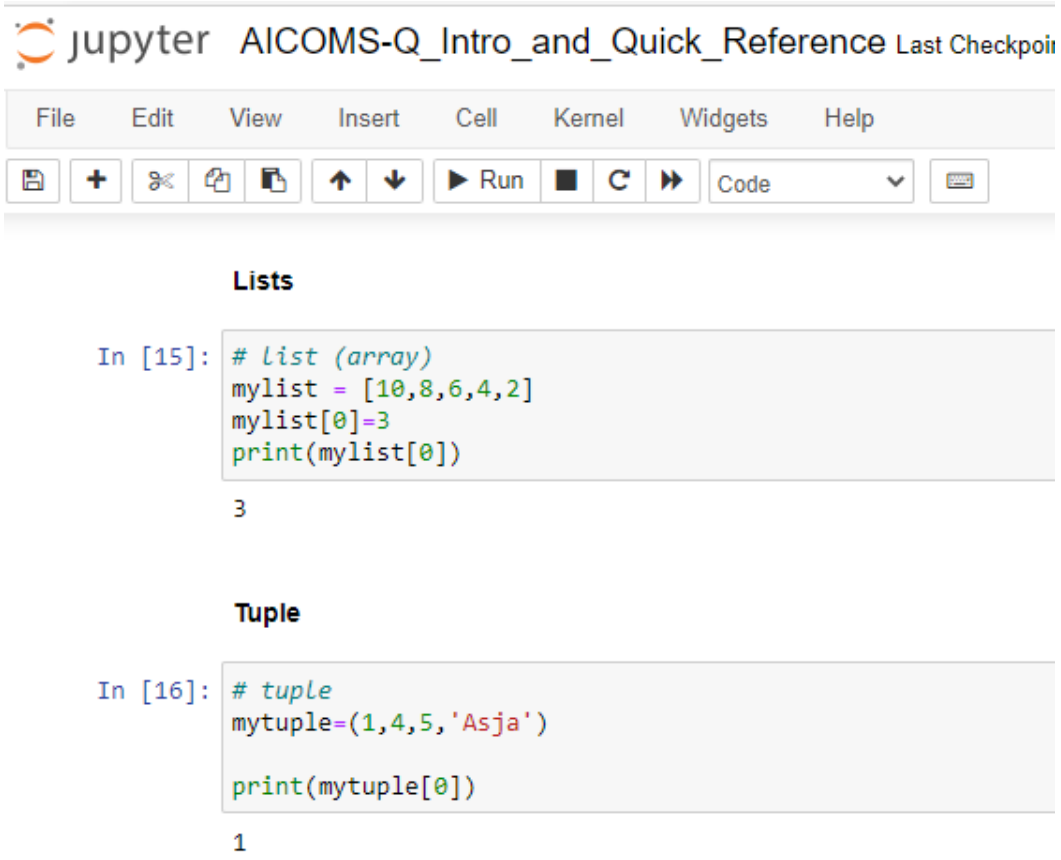
```
In [34]: data1 = 10   #example
         _data_1 = 10  #it can contain underscores
         data$ = 5
         break = 1
```

```
  File "<ipython-input-34-670bb89f3a2b>", line 3
    data$ = 5
         ^
SyntaxError: invalid syntax
```

- Names are case sensitive and cannot start with a number.
- They can contain letters (a-z), numbers (0-9) and underscores (_)
  name name1 _name name_1
- Symbols cannot be used
- There are some reserved words :

  and assert break class continue def del elif else except exec finally for from global if import in is lambda not or pass print raise return try while

The Basics [5/7]

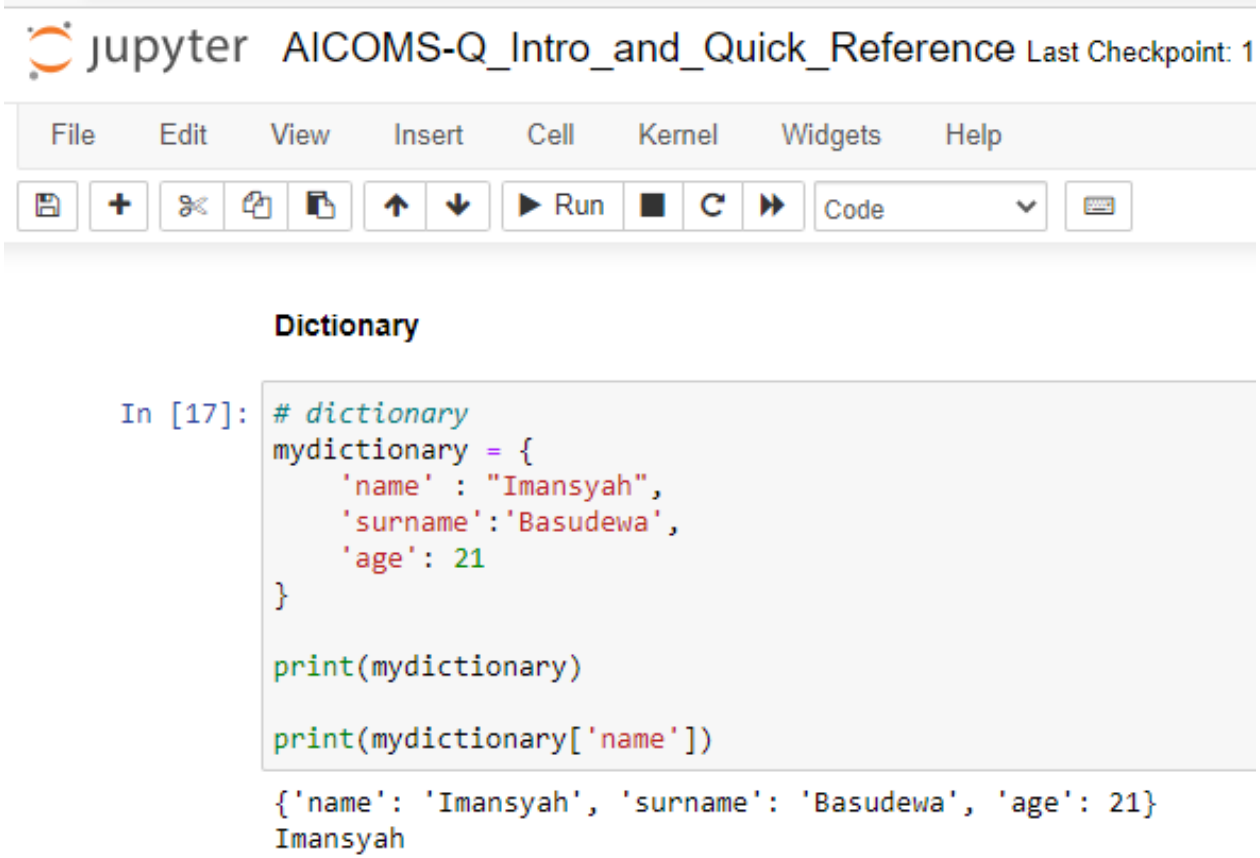


- **Positive and negative indices**
- Positive index : count from left, starting with 0
- Negative index : count from right, starting with -1

- **Sequence Types**
- List is a mutable ordered sequences of items of mixed types
- Lists are defined using square brackets (and commas)
- Tuple is a simple immutable ordered sequence of items
- Tuples are defined using paranthesis

- **Slicing and Copying**
- Slicing : return a copy of subset
- Return a copy of a container with a subset of the original members.
  t [1:4]
- Start copying at the first index and stop copying before second

The Basics [6/7]



- **Sequence Types**
- Dictionaries are data structures in which each particular value is associated with a particular label
- Data collected in the dictionary has no internal order but only the definition of a key-value pair

The Basics [7/7]

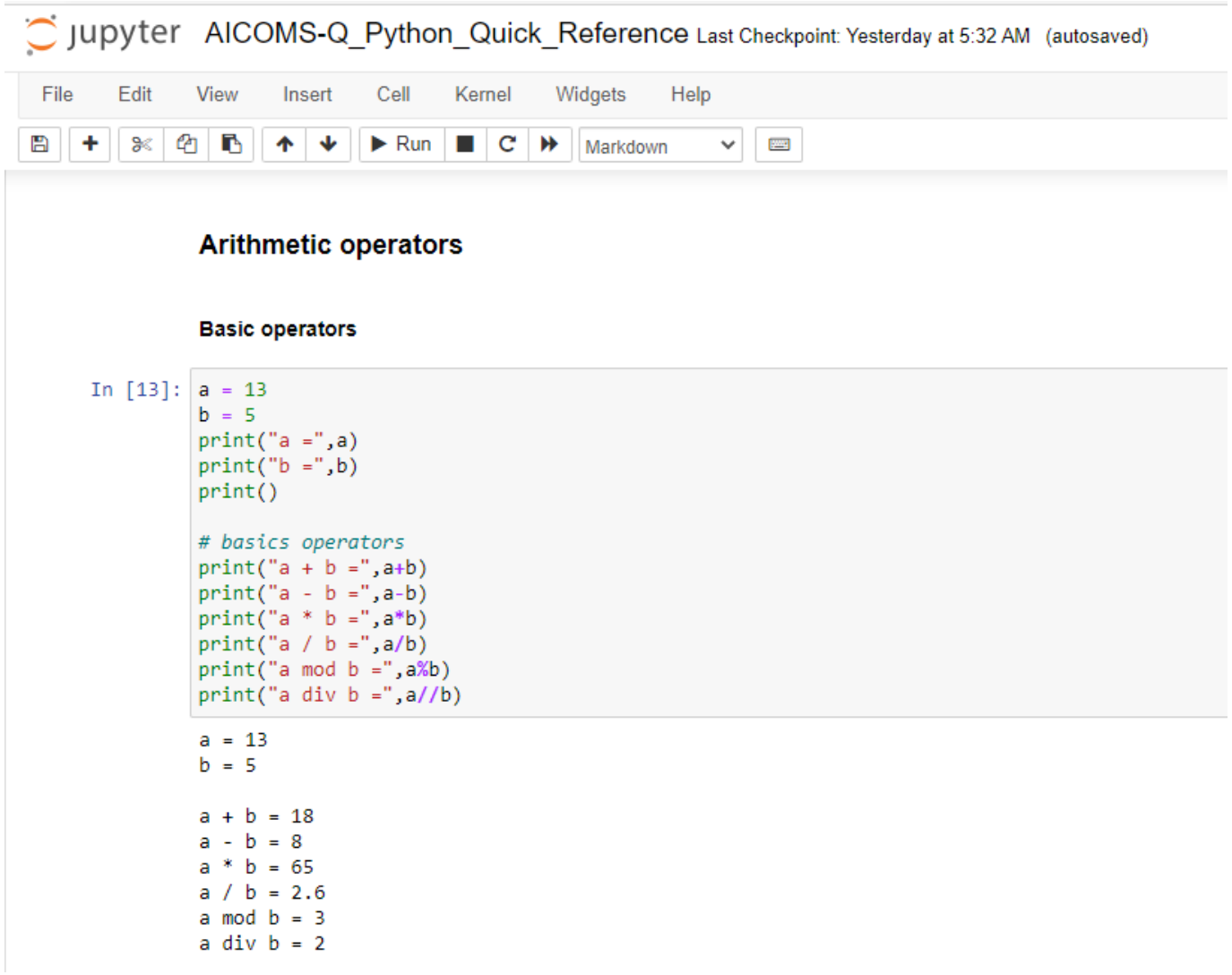
List of the other objects or variables

In [18]:
```
# list of the other objects or variables
list_of_other_objects =[
    mylist,
    mytuple,
    3,
    "Ada",
    mydictionary
]

print(list_of_other_objects)
print()
for el in list_of_other_objects:
    print(el)
```

```
[[3, 8, 6, 4, 2], (1, 4, 5, 'Asja'), 3, 'Ada', {'name': 'Imansyah', 'surname': 'Basudewa', 'age': 21}]

[3, 8, 6, 4, 2]
(1, 4, 5, 'Asja')
3
Ada
{'name': 'Imansyah', 'surname': 'Basudewa', 'age': 21}
```

- **List of the other objects or variables**
- List can contains list, tuple, dictionary, numbers, strings

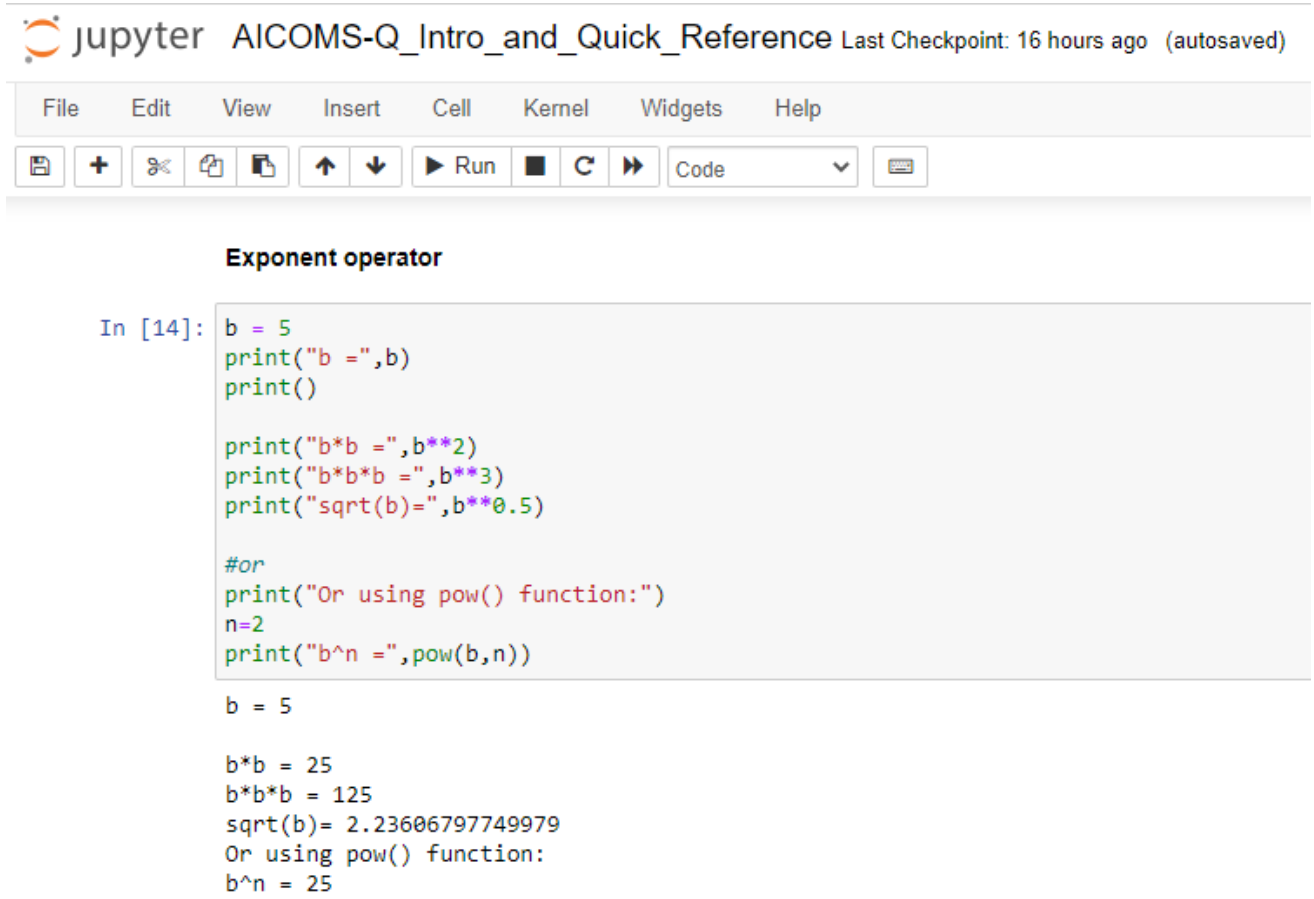
Operator [1/3]



Basic Arithmatics Operator

- Addition (+)
- Substraction (-)
- Multiplication (*)
- Division (/ or div)
- Power (** or pow())
- Modulo (%)

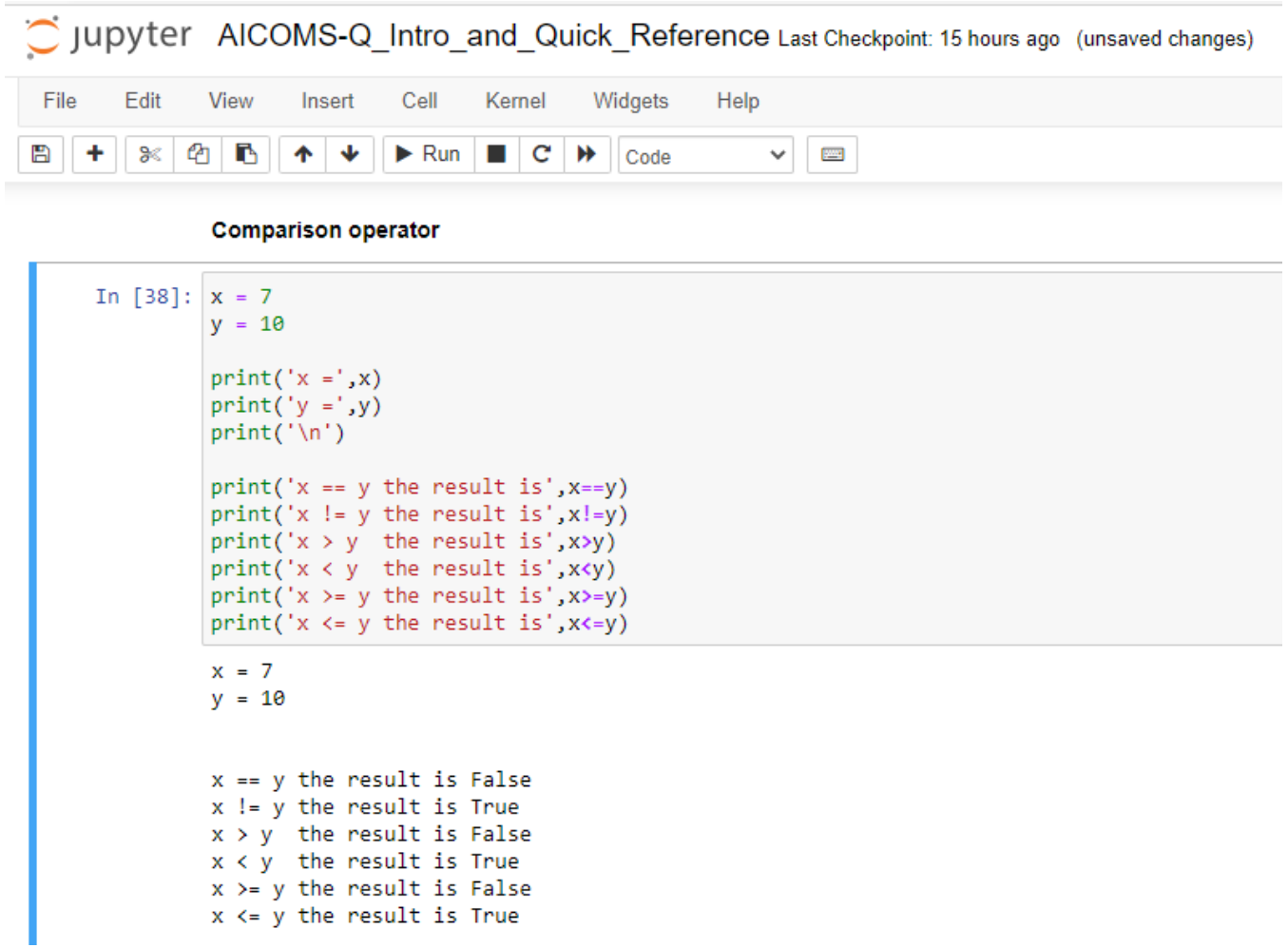
Operator [2/3]



Exponent operator

- We can use **
- Or using pow function pow()

Operator [3/3]



Comparison operator

- == equal to
- != not equal to
- > more than
- < less than
- >= more than equal to
- <= less than equal to

Loops and Functions [1/3]

While-loop

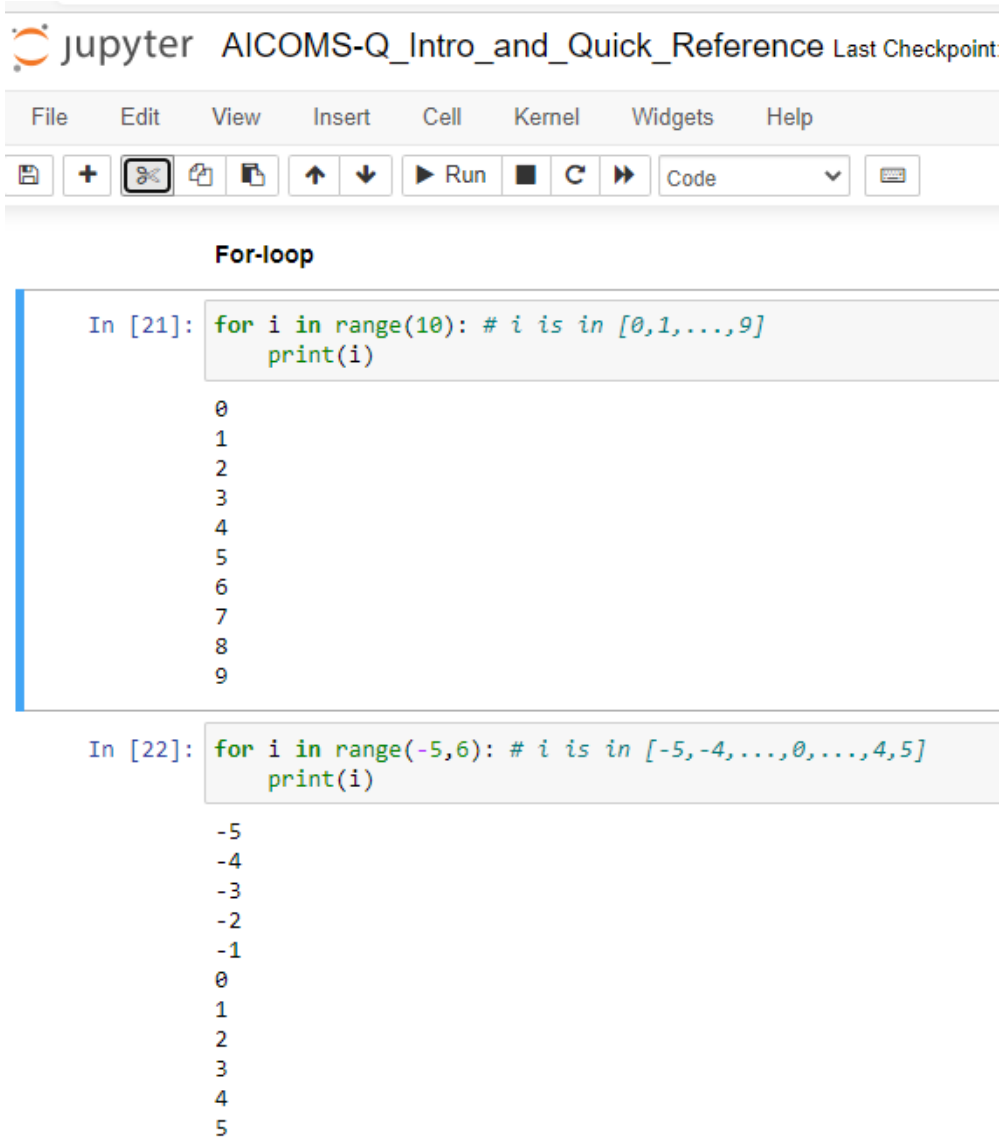
In [20]:
```
i = 10
while i>0: # while condition(s):
    print(i)
    i = i - 1
print("check")
```

```
10
9
8
7
6
5
4
3
2
1
check
```

While-loop

- With the while loop we can execute a set of statements as long as a condition is true.

Loops and Functions [2/3]

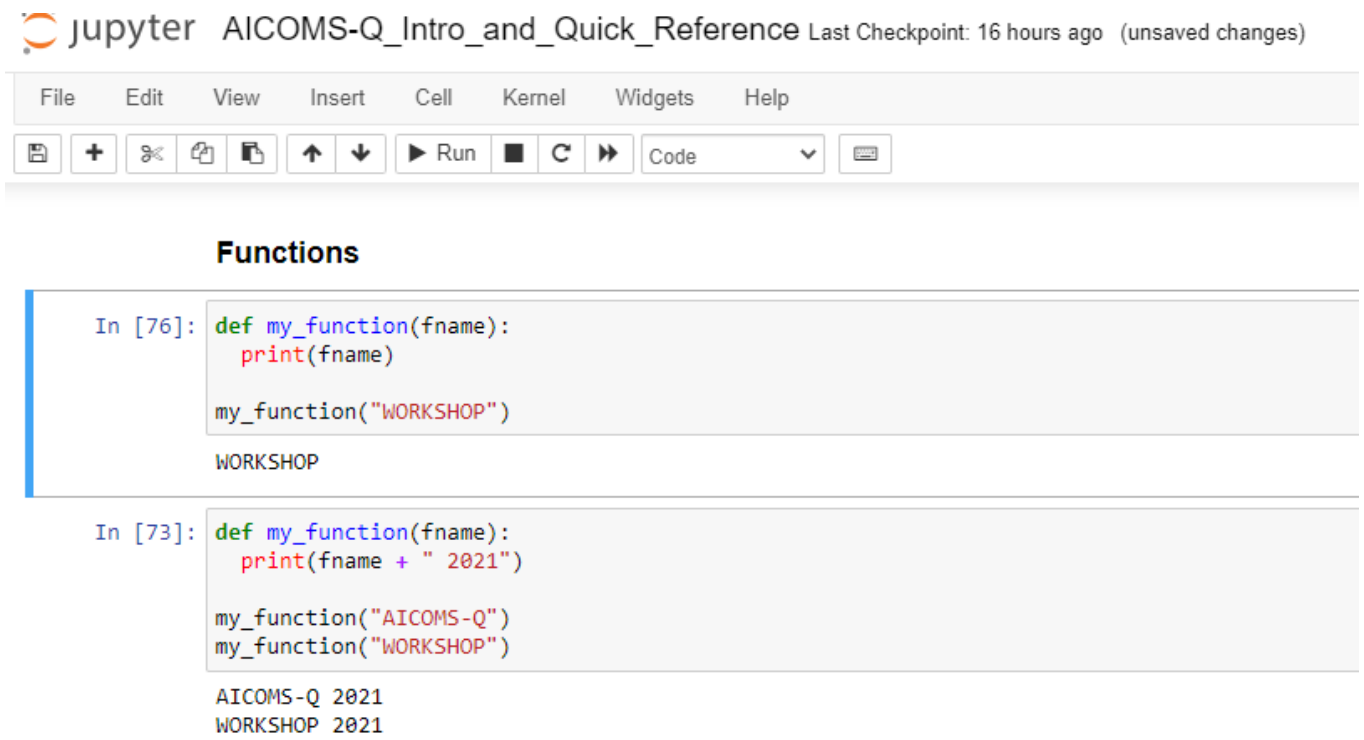


A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

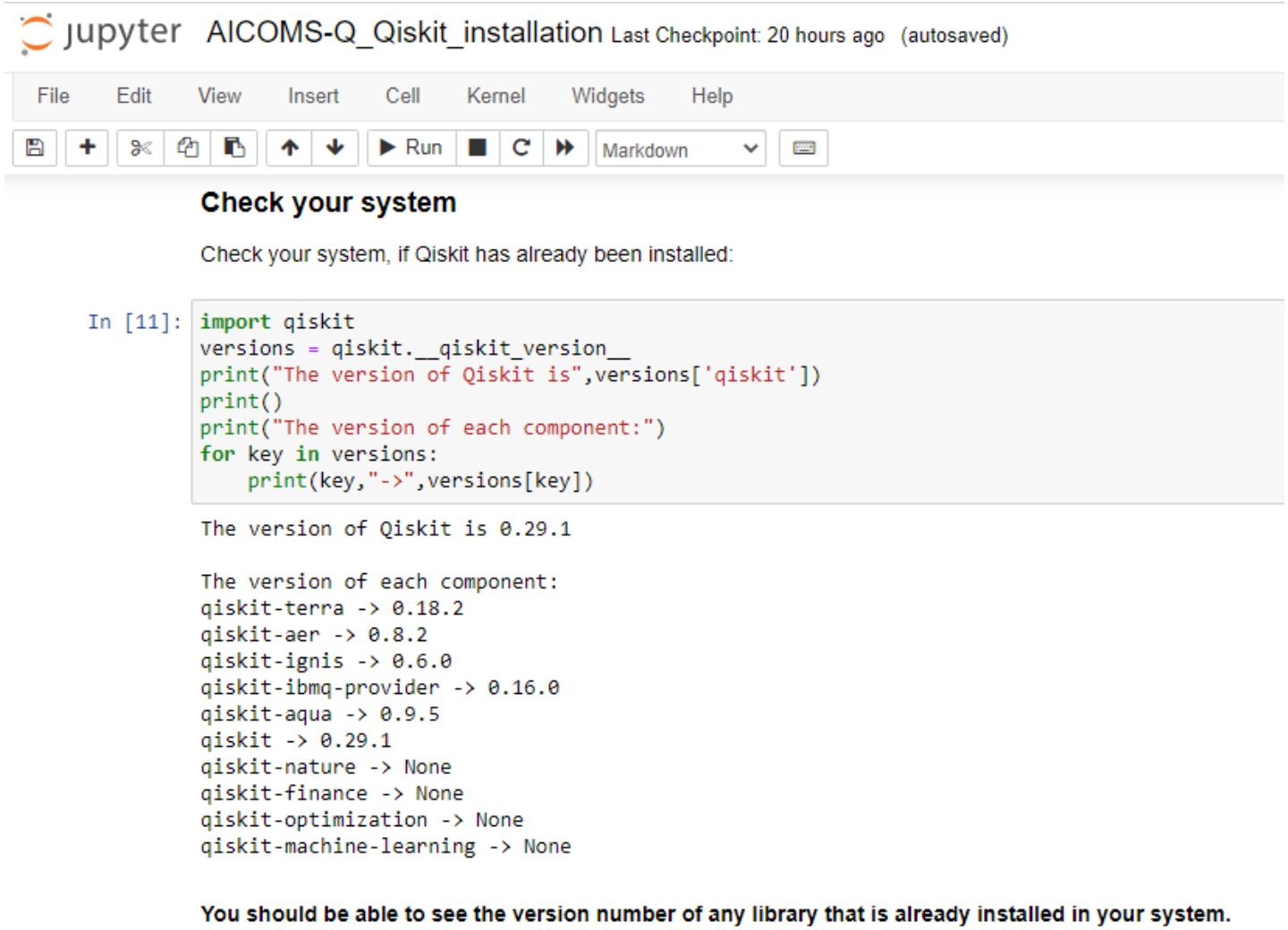
Loops and Functions [3/3]



Functions

- The advantage of function is that it can be used repeatedly with a variety of different values.
- The function more useful when it contains complex calculation

Library [1/5]

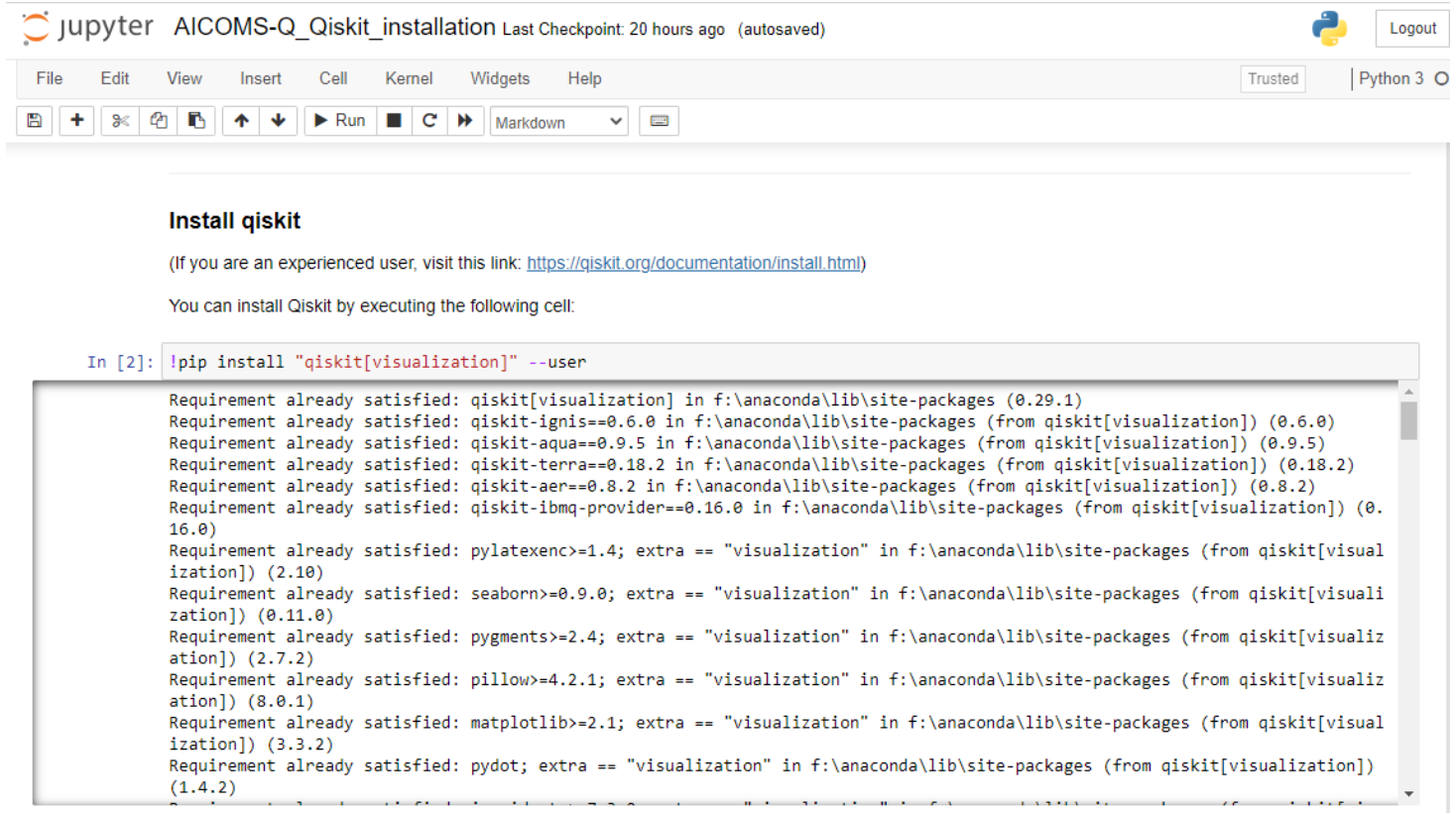


- Check the system
- Check the qiskit version with versions = qiskit.__qiskit_version_

Instalasi qiskit

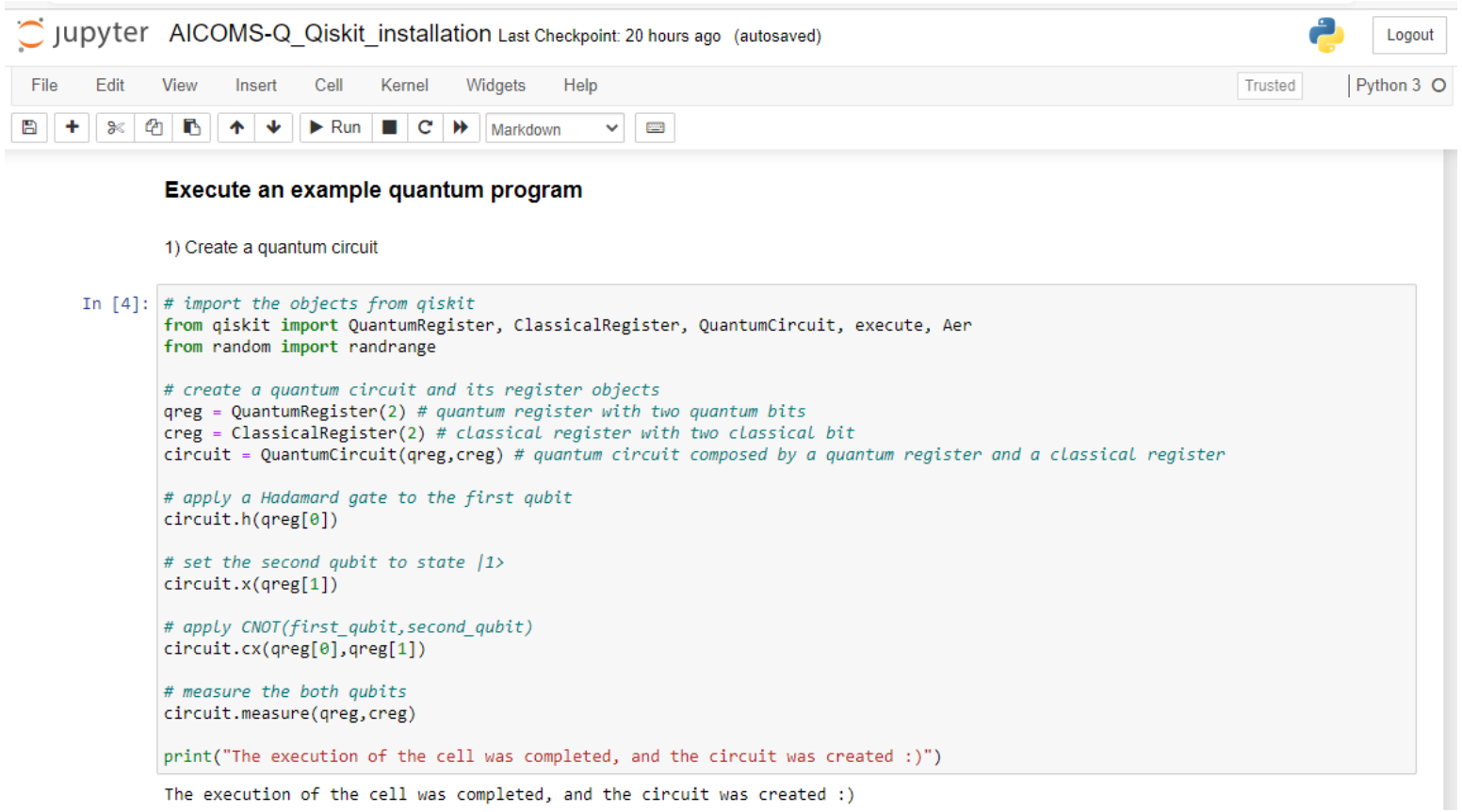
Qiskit Textbook

Library [2/5]



- Install qiskit by using following syntax
- !pip install “qiskit[visualization]” –user
- Run the code in the cell

Library [3/5]



- Qiskit contains Quantum register, Classical register, Quantum Circuit, etc

  Circuit library

Library [4/5]

jupyter AICOMS-Q_Qiskit_installation Last Checkpoint: 20 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

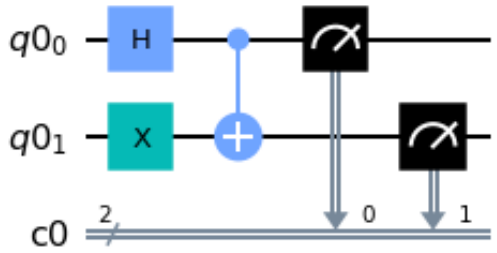
Run Markdown

2) Draw the circuit

Run the cell once more if the figure is not shown

In [5]:
```
# draw circuit
circuit.draw(output='mpl')

# the output will be a "matplotlib.Figure" object
```

Out[5]:



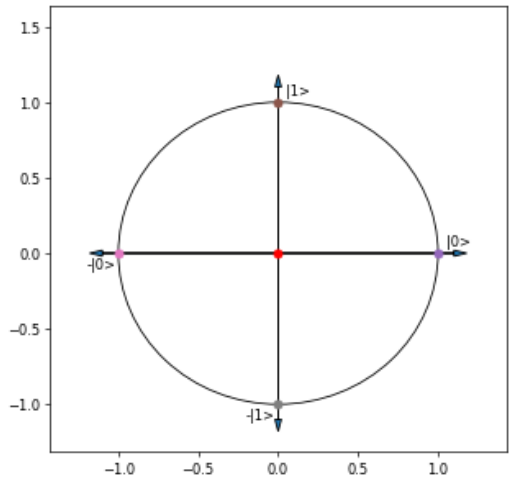
- Draw the circuit by running the code in the cell

Library [5/5]

In [9]:
```
import matplotlib
def draw_qubit():
    # draw a figure
    matplotlib.pyplot.figure(figsize=(6,6), dpi=60)
    # draw the origin
    matplotlib.pyplot.plot(0,0,'ro') # a point in red color
    # drawing the axes by using one of our predefined functions
    draw_axes()
    # drawing the unit circle by using one of our predefined functions
    draw_unit_circle()
    # drawing |0>
    matplotlib.pyplot.plot(1,0,"o")
    matplotlib.pyplot.text(1.05,0.05,"|0>")
    # drawing |1>
    matplotlib.pyplot.plot(0,1,"o")
    matplotlib.pyplot.text(0.05,1.05,"|1>")
    # drawing -|0>
    matplotlib.pyplot.plot(-1,0,"o")
    matplotlib.pyplot.text(-1.2,-0.1,"-|0>")
    # drawing -|1>
    matplotlib.pyplot.plot(0,-1,"o")
    matplotlib.pyplot.text(-0.2,-1.1,"-|1>")
```

In [10]:
```
draw_qubit()
```

- draw_axes() and draw_unit circle() are functions
- Show functions by using syntax *draw_qubit()*



https://qiskit.org/documentation/intro_tutorial1.html