Computer Programming I

Binnur Kurt, PhD

BAU
Bahçeşehir University

binnur.kurt@rc.bau.edu.tr

MODULE 3

INTRODUCTION TO PYTHON

Python

- Python is an object-oriented scripting language that was released publicly in 1991.
- It was developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam.
- Name is inspired by Monty Python a comedy show on BBC 1970s
- Python has rapidly become one of the world’s most popular programming languages.
- It’s now particularly popular for educational and scientific computing, and it recently surpassed the programming language R as the most popular data-science programming language.

Python

- Here are some reasons why Python is popular
  - It’s open source, free and widely available with a massive open-source community
  - It’s easier to learn than languages like C, C++, C# and Java, enabling novices and professional developers to get up to speed quickly
  - It’s easier to read than many other popular programming languages
  - It’s widely used in education
  - It enhances developer productivity with extensive standard libraries and thousands of third-party open-source libraries, so programmers can write code faster and perform complex tasks with minimal code
  - There are massive numbers of free open-source Python applications
  - It’s popular in web development: Django, Flask, …

Python

> Here are some reasons why Python is popular

- It supports popular programming paradigms
  - Procedural
  - Functional
  - Object-oriented
  - Reflective
- It simplifies concurrent programming
  - asyncio and async/await
- There are lots of capabilities for enhancing Python performance
- It's used to build anything from simple scripts to complex apps with massive numbers of users, such as Dropbox, YouTube, Reddit, Instagram and Quora

Python

> Here are some reasons why Python is popular

- It's popular in artificial intelligence, which is enjoying explosive growth, in part because of its special relationship with data science
- It's widely used in the financial community
- There's an extensive job market for Python programmers across many disciplines, especially in data-science-oriented positions

Philosophy

> PEP 20, The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.

Philosophy

> PEP 20, The Zen of Python

- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

import this

```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\dcl>python
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

It’s the Libraries!

> Libraries help you avoid “reinventing the wheel”

> The Python Standard Library provides rich capabilities for

- text/binary data processing, mathematics,
- functional-style programming, file/directory access,
- data persistence, data compression/archiving,
- cryptography, operating-system services,
- concurrent programming, IPC, networking protocols,
- JSON/XML/other Internet data formats,
- multimedia,
- internationalization,
- GUI, debugging, profiling, . . .

Python Standard Library modules

- **collections**—Additional data structures beyond lists, tuples, dictionaries and sets.
- **csv**—Processing comma-separated value files.
- **datetime**, **time**—Date and time manipulations.
- **decimal**—Fixed-point and floating-point arithmetic, including monetary calculations.
- **doctest**—Simple unit testing via validation tests and expected results embedded in docstrings.
- **json**—JavaScript Object Notation (JSON) processing for use with web services and NoSQL document databases.
- **math**—Common math constants and operations.

Python Standard Library modules

- **os**—Interacting with the operating system.
- **timeit**—Performance analysis.
- **queue**—First-in, first-out data structure.
- **random**—Pseudorandom numbers.
- **re**—Regular expressions for pattern matching.
- **sqlite3**—SQLite relational database access.
- **statistics**—Mathematical statistics functions like mean, median, mode and variance.
- **string**—String processing.
- **sys**—Command-line argument processing; standard input, standard output and standard error streams.

Popular Python libraries used in data science

> ***Scientific Computing and Statistics***

– **NumPy** (Numerical Python)
  - NumPy provides efficient ndarray data structure to represent lists and matrices, and it also provides routines for processing such data structures.

– **SciPy** (Scientific Python)
  - Built on NumPy, SciPy adds routines for scientific processing, such as integrals, differential equations, additional matrix processing and more.
  - scipy.org controls SciPy and NumPy.

– **StatsModels**
  - Provides support for estimations of statistical models, statistical tests and statistical data exploration.

Popular Python libraries used in data science

> ***Data Manipulation and Analysis***

– **Pandas**
  - An extremely popular library for data manipulations.
  - Pandas makes abundant use of NumPy's **`ndarray`**
  - Its two key data structures are
    - –**`Series`** (*one dimensional*)
    - –**`DataFrames`** (two dimensional).

Popular Python libraries used in data science

> ***Visualization***

- Matplotlib
  - A highly customizable visualization and plotting library.
  - Supported plots include regular, scatter, bar, contour, pie, quiver, grid, polar axis, 3D and text.
- Seaborn
  - A higher-level visualization library built on Matplotlib.
  - Seaborn adds a nicer look-and-feel, additional visualizations and enables you to create visualizations with less code.

Popular Python libraries used in data science

> ***Machine Learning, Deep Learning and Reinforcement Learning***

- scikit-learn
  - Top machine-learning library
  - Machine learning is a subset of AI. Deep learning is a subset of machine learning that focuses on neural networks.
- Keras
  - One of the easiest to use deep-learning libraries.
  - Keras runs on top of TensorFlow (Google), CNTK (Microsoft's cognitive toolkit for deep learning) or Theano (Université de Montréal).

Popular Python libraries used in data science

> ***Machine Learning, Deep Learning and Reinforcement Learning***

– **TensorFlow**
  - The most widely used deep learning library from Google
  - TensorFlow works with
    – GPUs (graphics processing units)
    – Google's custom TPUs (Tensor processing units) for performance.
  - TensorFlow is important in AI and big data analytics
    – Processing demands are enormous.

Popular Python libraries used in data science

> ***Natural Language Processing (NLP)***

– **NLTK** *(Natural Language Toolkit)*
  - A highly customizable visualization and plotting library.
  - Supported plots include regular, scatter, bar, contour, pie, quiver, grid, polar axis, 3D and text.

– **TextBlob**
  - An object-oriented NLP text-processing library built on the NLTK and pattern NLP libraries.
  - Simplifies many NLP tasks.

– **Gensim** *(Similar to NLTK)*
  - Commonly used to build an index for a collection of documents, then determine how similar another document is to each of those in the index.

PYTHON AND JUPYTER INSTALLATION ON LINUX

> First check RHEL already have an older Python

```
# python -V
```

> Copy new Python zip file under / and extract

> Follow README to install Python

```
yum install zlib-devel openssl-devel
./configure; make; make test; make install
```

> Install ipython and jupyter using pip

```
# pip list; pip install --upgrade pip
# pip install ipython
# pip install jupyter
```

INSTALLING ANACONDA AND PYTHON FOR WINDOWS

What Is Anaconda?

> Anaconda is a Python and R distribution software.
> It aims to provide everything you need for Python "out of the box."
> Its primary use is for data analytics and data science; however, it's a superb tool for learning as well.
> Anconda includes
  – The core Python language and libraries
  – Jupyter Notebook
  – Anaconda's own package manager

What Is Jupyter Notebook?

> It is an open-source integrated development environment (IDE)
> It allows you to create and share documents that contain live code, equations, visualizations, and narrative text.
> It also allows you to write snippets of code without needing to know a lot about Python.

Installing Anaconda and Python for Windows

> For Windows users, Python usually isn't included, but it gets installed with Anaconda.
> Use the following steps to install Anaconda properly:

1. Open your browser and type
    - ***https://www.anaconda.com/products/individual***
2. Click the download button
3. Once you are on the next page, make sure you select the proper operating system and then click that button

Installing Anaconda and Python for Windows

anaconda.com/products/individual

Individual Edition

Your data science toolkit

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Download

Installing Anaconda and Python for Windows

Anaconda Installers

Windows	MacOS	Linux
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (466 MB)	64-Bit Graphical Installer (462 MB)	64-Bit (x86) Installer (550 MB)
32-Bit Graphical Installer (397 MB)	64-Bit Command Line Installer (454 MB)	64-Bit (Power8 and Power9) Installer (290 MB)

Checking a Version Number

> The terminal is always a great way to check version numbers

```
python --version
```

Using the Python Shell

> Python is a language that requires what is called an “interpreter” to read and run the code we create.

> When the Python shell is activated, it acts as a local interpreter within the terminal session that is open.

> While it’s open, we can write any Python that we wish to execute.

> This is generally great for practicing small snippets of code, so that you don’t have to open an IDE and run an entire file.

> To start the Python shell up, simply type “python” and hit enter

```
D:\DEVEL\stage\tmp\dcl160>python
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Writing Your First Line of Python

```
>>> print("Hello Mars!")
Hello Mars!
>>> exit()
```

IPython Basics

> You can launch the IPython shell on the command line just like launching the regular Python interpreter:

```
ipython
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC
v.1916 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.5.0 -- An enhanced Interactive Python. Type '?' for
help.

In [1]:
```

ipython

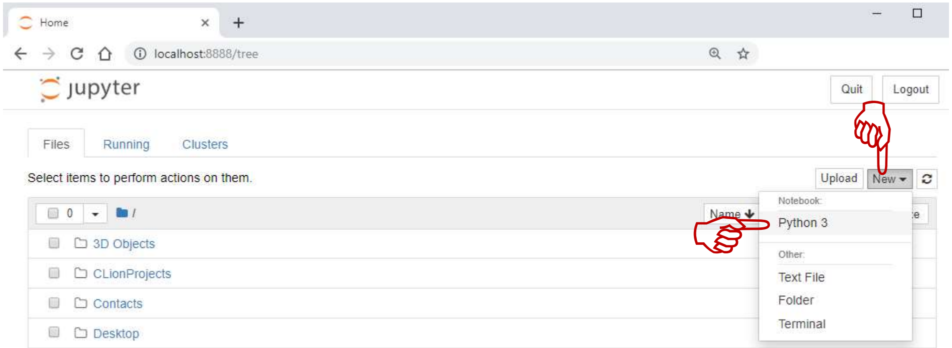
```
In [5]: import numpy as np

In [6]: data = {i : np.random.randn() for i in range(7)}

In [7]: data
Out[7]:
{0: -0.20470765948471295,
 1: 0.47894333805754824,
 2: -0.5194387150567381,
 3: -0.55573030434749,
 4: 1.9657805725027142,
 5: 1.3934058329729904,
 6: 0.09290787674371767}
```

Opening Jupyter Notebook

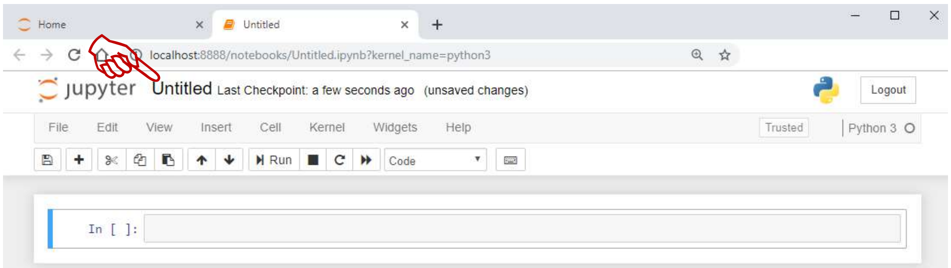
- Jupyter Notebook can be opened through the Anaconda program
- Jupyter Notebook cab opened through the terminal
  - Jupyter Notebook will open in the same directory that our terminal is in
  - Knowing how to use terminal will help you as a developer
  - If you still have the terminal session from yesterday open, skip the first step

```
$ jupyter notebook
```

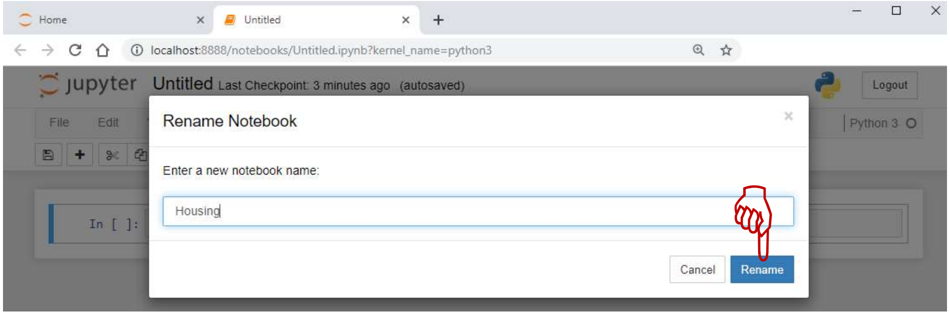
Running Jupyter



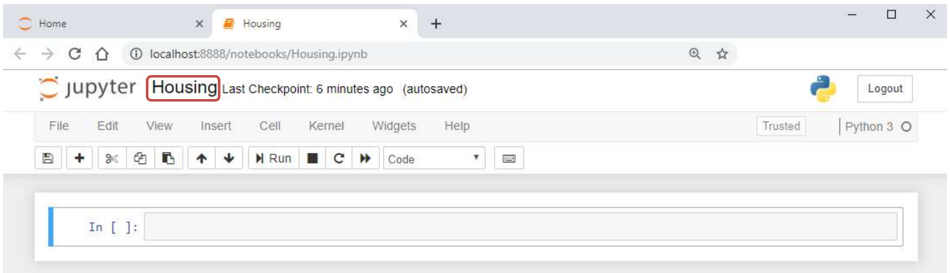
Running Jupyter



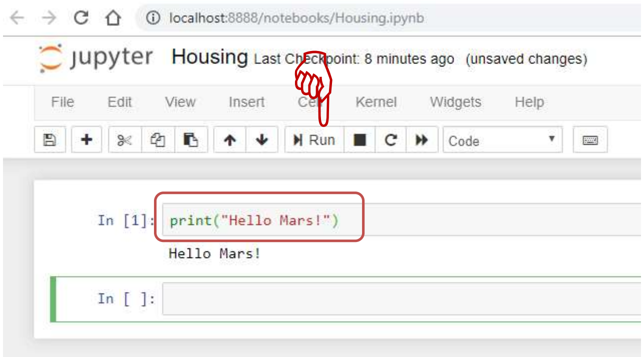
Running Jupyter



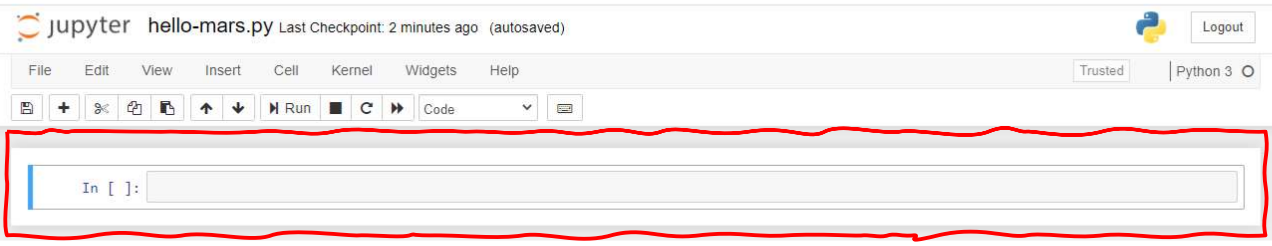
Running Jupyter



Running Jupyter



Jupyter Notebook Cells



TYPES

Comments

```
# this is a comment
" " "
      This is a multi-Line comment
" " "
print("Hello") # this is also a comment
```

What Are Data Types?

Type	Description
None	The Python “null” value (only one instance of the None object exists)
str	String type; holds Unicoded (UTF-8 encoded) strings
bytes	Raw ASCII bytes (or Unicode encoded as bytes)
float	Double-precision (64-bit) floating-point number
bool	A true or False value
int	Arbitrary precision signed integer

Type System

```
In [2]: type(3.1415)
Out[2]: float

In [3]: type(42)
Out[3]: int

In [4]: type('Jack Bauer')
Out[4]: str

In [5]: type(3.)
Out[5]: float

In [6]: type("Kate Austen")
Out[6]: str

In [7]: type(None)
Out[7]: NoneType

In [8]: type(False)
Out[8]: bool
```

Type System

```
In [9]: type((0.1+0.2)+0.3 == 0.1 + (0.2+0.3))
Out[9]: bool

In [10]: (0.1+0.2)+0.3 == 0.1 + (0.2+0.3)
Out[10]: False

In [11]: 100 * 4.35
Out[11]: 434.99999999999994

In [12]: 2.0 - 1.1
Out[12]: 0.8999999999999999
```

Numeric types

> The primary Python types for numbers are int and float.

> An int can store arbitrarily large numbers:

```
In [48]: ival = 17239871

In [49]: ival ** 6
Out[49]: 26254519291092456596965462913230729701102721
```

Numeric types

- Floating-point numbers are represented with the Python float type.
  - A double-precision (64-bit) value.
- They can also be expressed with scientific notation:

```
In [50]: fval = 7.243

In [51]: fval2 = 6.78e-5
```

- Integer division not resulting in a whole number will always yield a floating-point number:

```
In [52]: 3 / 2
Out[52]: 1.5
```

Numeric types

- To get C-style integer division, use the floor division operator **//**:
- It drops the fractional part if the result is not a whole number

```
In [53]: 3 // 2
Out[53]: 1
```

Arithmetic Operators

Python Operation	Arithmetic Operator	Algebraic Expression	Python Expression
Addition	+	f+7	f + 7
Subtraction	-	p-c	p – c
Multiplication	*	b*m	b * m
Exponentiation	**	x^y	x ** y
True division	/	x/y or $\frac{x}{y}$ or x ÷ y	x / y
Floor division	//	$\lfloor x/y \rfloor$ or $\left\lfloor \frac{x}{y} \right\rfloor$ or $\lfloor$x ÷ y$\rfloor$	x // y
Remainder (modulo)	%	r mod s	r % s

Exceptions

> Dividing by zero with / or // is not allowed and results in an exception

```
In [30]: x = 2.0

In [31]: y = x / 0.0

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-31-0332a55bc4af> in <module>
----> 1 y = x / 0.0

ZeroDivisionError: float division by zero
```

Operator Precedence Rules

> Rules of operator precedence

1. Expressions in parentheses evaluate first, so parentheses may force the order of evaluation to occur in any sequence you desire.
   – Parentheses have the highest level of precedence.
   – In expressions with nested parentheses, such as (a / (b - c)), the expression in the innermost parentheses (that is, b - c) evaluates first.
2. Exponentiation operations evaluate next. If an expression contains several exponentiation operations, Python applies them from right to left.
3. Multiplication, division and modulus operations evaluate next.
   – If an expression contains several multiplication, true-division, floor-division and modulus operations, Python applies them from left to right.
   – Multiplication, division and modulus are "on the same level of precedence."

Operator Precedence Rules

> Rules of operator precedence

4. Addition and subtraction operations evaluate last.
   – If an expression contains several addition and subtraction operations, Python applies them from left to right.
   – Addition and subtraction also have the same level of precedence.

Type casting

> The **`str`**, **`bool`**, **`int`**, and **`float`** types are also functions that can be used to cast values to those types:

```
In [91]: s = '3.14159'

In [92]: fval = float(s)

In [93]: type(fval)
Out[93]: float

In [94]: int(fval)
Out[94]: 3

In [95]: bool(fval)
Out[95]: True

In [96]: bool(0)
Out[96]: False
```

None

> **`None`** is the Python null value type.

> If a function does not explicitly return a value, it implicitly returns **`None`**:

```
In [97]: a = None

In [98]: a is None
Out[98]: True

In [99]: b = 5

In [100]: b is not None
Out[100]: True
```

None

> **None** is also a common default value for function arguments:

```
def add_and_maybe_multiply(a, b, c=None):
    result = a + b

    if c is not None:
        result = result * c

    return result
```

> While a technical point, it's worth bearing in mind that **None** is not only a reserved keyword but also a unique instance of **NoneType**:

```
In [101]: type(None)
Out[101]: NoneType
```

Complex Numbers

```
In [41]: c = complex(1,2)

In [42]: print(c.real)
1.0

In [53]: print(c.imag)
2.0

In [52]: print(c)
(1+2j)

In [55]: print(c.conjugate())
(1-2j)

In [58]: print(c * c.conjugate())
(5+0j)

In [59]: print((c * c.conjugate()).real)
5.0
```

Strings and Quotes

```
In [7]: "This is a string using a double quote"
Out[7]: 'This is a string using a double quote'

In [8]:  'This is a string with a single quote'
Out[8]: 'This is a string with a single quote'

In [9]: """This string has three quotes
        look at
        what it can do!"""
Out[9]: 'This string has three quotes\nlook at \nwhat it can do!'
```

Using the print() Function

> The `print()` function is used whenever you want to print text to the screen

```
In [5]: print("This is a string using a double quote")
        This is a string using a double quote

In [3]: print('This is a string with a single quote')
Out[3]: 'This is a string with a single quote'

In [6]: print("""This string has three quotes
        look at
        what it can do!""")
        This string has three quotes
        look at
        what it can do!
```

Using the print() Function

```
In [25]: print('Welcome' 'to' 'Python!')

WelcometoPython!

In [26]: print('Welcome', 'to', 'Python!')

Welcome to Python!

In [31]: print("Employee [first name: %s, last name: %s, salary: %6.1f]"
               % ("Jack" , "Bauer", 123456.78))

Employee [first name: Jack, last name: Bauer, salary: 123456.8]
```

Using the print() Function

```
In [11]: print("I said, \"Don’t do it\"")

I said, "Don’t do it"

In [12]: print("""Roses are red
         Violets are blue
         I just printed multiples lines
         And you did too!""")

Roses are red
Violets are blue
I just printed multiples lines
And you did too!

In [14]: print("Roses are red \n Violets are blue \n \
         I just printed multiple \
         lines \n And you did too!")

Roses are red
 Violets are blue
 I just printed multiple lines
 And you did too!
```

Using the print() Function

```
In [15]: print(r"Roses are red \n Violets are blue \n \
         I just printed multiple \
         lines \n And you did too!")

         Roses are red \n Violets are blue \n \
         I just printed multiple \
         lines \n And you did too!
```

```
In [23]: print("pi=%3.6f" % 3.14159265359)

         pi=3.141593
```

Using the print() Function

```
# This prints out: A list: [1, 2, 3]
mylist = [1,2,3]
print("A list: %s" % mylist)
```

Escape Sequences

Escape sequence	Description
`\n`	Insert a newline character in a string. When the string is displayed, for each newline, move the screen cursor to the beginning of the next line.
`\t`	Insert a horizontal tab. When the string is displayed, for each tab, move the screen cursor to the next tab stop.
`\\`	Insert a backslash character in a string.
`\”`	Insert a double quote character in a string.
`\’`	Insert a single quote character in a string.

Number-theoretic and representation functions

> `math.ceil(x)`

- Returns the ceiling of x, the smallest integer greater than or equal to x. If x is not a float, delegates to x.__ceil__(), which should return an Integral value.

> `math.comb(n, k)`

- Returns the number of ways to choose k items from n items without repetition and without order.
- Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates to zero when k > n.
- Also called the binomial coefficient because it is equivalent to the coefficient of k-th term in polynomial expansion of the expression (1 + x) ** n.
- Raises TypeError if either of the arguments are not integers. Raises ValueError if either of the arguments are negative.

Number-theoretic and representation functions

> `math.copysign(x, y)`

– Returns a float with the magnitude (absolute value) of x but the sign of y. On platforms that support signed zeros, copysign(1.0, -0.0) returns -1.0.

> `math.fabs(x)`

– Returns the absolute value of x.

> `math.factorial(x)`

– Returns x factorial as an integer. Raises ValueError if x is not integral or is negative.

> `math.floor(x)`

– Returns the floor of x, the largest integer less than or equal to x. If x is not a float, delegates to x.__floor__(), which should return an Integral value.

Power and logarithmic functions

> `math.exp(x)`

– Return e raised to the power x, where e = 2.718281… is the base of natural logarithms.

– This is usually more accurate than math.e ** x or pow(math.e, x)

> `math.expm1(x)`

– Return e raised to the power x, minus 1.

– Here e is the base of natural logarithms.

– For small floats x, the subtraction in exp(x) - 1 can result in a significant loss of precision; the expm1() function provides a way to compute this quantity to full precision

Power and logarithmic functions

```
In [104]: from math import exp, expm1

In [108]: exp(1e-5) - 1  # gives result accurate to 11 places
Out[108]: 1.0000050000069649e-05

In [107]: expm1(1e-5) # result accurate to full precision
Out[107]: 1.0000050000166667e-05
```

Power and logarithmic functions

> `math.log(x[, base])`
  - With one argument, returns the natural logarithm of x (to base e).
  - With two arguments, returns the logarithm of x to the given base, calculated as log(x)/log(base).

> `math.log1p(x)`
  - Returns the natural logarithm of 1+x (base e).
  - The result is calculated in a way which is accurate for x near zero.

> `math.log2(x)`
  - Returns the base-2 logarithm of x.
  - This is usually more accurate than log(x, 2).

Power and logarithmic functions

> `math.log10(x)`
  – Returns the base-10 logarithm of x.
  – This is usually more accurate than log(x, 10).

> `math.pow(x, y)`
  – Returns x raised to the power y.
  – Unlike the built-in ** operator, math.pow() converts both its arguments to type float.
  – Use ** or the built-in pow() function for computing exact integer powers.

> `math.sqrt(x)`
  – Returns the square root of x.

Trigonometric functions

> `math.acos(x)`
  – Returns the arc cosine of x, in radians.

> `math.asin(x)`
  – Returns the arc sine of x, in radians.

> `math.atan(x)`
  – Returns the arc tangent of x, in radians.

> `math.atan2(y, x)`
  – Returns atan(y / x), in radians.
  – The result is between -pi and pi.

> `math.cos(x)`
  – Returns the cosine of x radians.

Trigonometric functions

> `math.sin(x)`
  - Returns the sine of x radians.

> `math.tan(x)`
  - Returns the tangent of x radians.

> `math.dist(p, q)`
  - Returns the Euclidean distance between two points p and q, each given as a sequence (or iterable) of coordinates.
  - The two points must have the same dimension.

Angular conversion

> `math.degrees(x)`
  - Converts angle x from radians to degrees.

> `math.radians(x)`
  - Converts angle x from degrees to radians

Hyperbolic functions

> Hyperbolic functions are analogs of trigonometric functions that are based on hyperbolas instead of circles.

> `math.acosh(x)`
  – Returns the inverse hyperbolic cosine of x.

> `math.asinh(x)`
  – Returns the inverse hyperbolic sine of x.

> `math.atanh(x)`
  – Returns the inverse hyperbolic tangent of x.

> `math.cosh(x)`
  – Returns the hyperbolic cosine of x.

> `math.sinh(x)`
  – Returns the hyperbolic sine of x.

Hyperbolic functions

> `math.tanh(x)`
  – Returns the hyperbolic tangent of x.

Special functions

> `math.erf(x)`

- Returns the error function at x.
- The erf() function can be used to compute traditional statistical functions such as the cumulative standard normal distribution

> `math.erfc(x)`

- Returns the complementary error function at x.
- The complementary error function is defined as 1.0 - erf(x).
- It is used for large values of x where a subtraction from one would cause a loss of significance.

> `math.gamma(x)`

- Returns the Gamma function at x.

Special functions

> `math.lgamma(x)`

- Returns the natural logarithm of the absolute value of the Gamma function at x.

Constants

> `math.pi`
  – The mathematical constant π = 3.141592…, to available precision.

> `math.e`
  – The mathematical constant e = 2.718281…, to available precision.

> `math.tau`
  – The mathematical constant τ = 6.283185…, to available precision.
  – Tau is a circle constant equal to 2π, the ratio of a circle’s circumference to its radius.

> `math.inf`
  – A floating-point positive infinity.
  – Equivalent to the output of float('inf').

Constants

> `math.nan`
  – A floating-point “not a number” (NaN) value.
  – Equivalent to the output of float('nan').

Examples

```
In [78]: import math

In [81]: 4 * math.atan(1)
Out[81]: 3.141592653589793

In [83]: math.factorial(10)
Out[83]: 3628800

In [84]: math.factorial(20)
Out[84]: 2432902008176640000

In [90]: math.gcd(3,5)
Out[90]: 1

In [95]: math.sqrt(16)
Out[95]: 4.0
```

Examples

```
In [96]: min(47, 95, 88, 73, 88, 84)
Out[96]: 47

In [97]: max(47, 95, 88, 73, 88, 84)
Out[97]: 95

In [100]: sum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
Out[100]: 0.9999999999999999

In [102]: math.fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
Out[102]: 1.0
```

Arbitrary Precision Arithmetic

```
In [61]: from mpmath import *

In [64]: mp.dps = 10; mp.pretty = True

In [65]: 4*atan(1)
Out[65]: 3.141592654

In [66]: mp.dps = 100; mp.pretty = True

In [67]: 4*atan(1)
Out[67]: 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706

In [71]: limit(lambda n: (1+1/n)**n, inf)
Out[71]: 2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642
```