Introduction to Python

David Pinelle, Ph.D., Computation Research Manager
Social Sciences Research Laboratories
University of Saskatchewan
Saskatoon, Saskatchewan, Canada

ssrl.usask.ca

Assumptions

- Assumptions for this workshop
  - You do not need programming experience
  - You do not need experience with Python

ssrl.usask.ca

Agenda

- Introduction
- Running Python
- Python Programming
  - Data types (Numeric, Strings, Lists)
  - Control flow (If / then, While loops, For loops)
  - Functions
  - Modules
  - Numpy NDArray

Assignments

- Available on Github:
- https://github.com/pinelle

ssrl.usask.ca

University of Saskatchewan | SSRL

Why Python?

- Python is a general-purpose programming language: graphics, graphical user interface, web programming, etc.
- It's popular, so there are many learning resources online
- It is easy to use and learn
  - Simplified syntax with an emphasis on natural language
  - Allows focus on the problem rather than on the language
- Free and open source

Why Python?

- Python has strong numeric processing capabilities: matrix operations, etc.
- Suitable for machine learning , deep learning, visualization, and more.

Library	Usage
numpy, scipy	Scientific & technical computing
pandas	Data manipulation and aggregation
scikit-learn	Machine learning
tensorflow	Deep learning
statsmodel	Statistical analysis
matplotlib, seaborn	Visualization
beautifulsoup, scrapy	Web scraping
nltk, gensim	Text processing

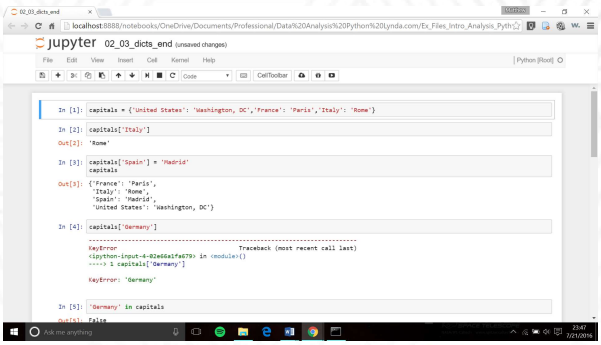
Python versions

- Python 2 (v. 2.7.x)
  - Old release but still available because Python 3 is not backwards compatible
- Python 3 (v. 3.7.x)
  - Recommended release for new projects
- Anaconda
  - Python 2 and 3 versions are available
  - Includes more than 1400 popular data-science packages + applications
    - Spyder
    - Jupyter Notebook

Integrated development environments (IDEs)

- Provides many features like coding, debugging, executing, autocompleting, libraries in one place

1. Spyder Scientific Python IDE
2. Jupyter Notebook
3. PyDev for Eclipse
4. PyCharm
5. Visual Studio Code
6. Idle



Technology for this workshop

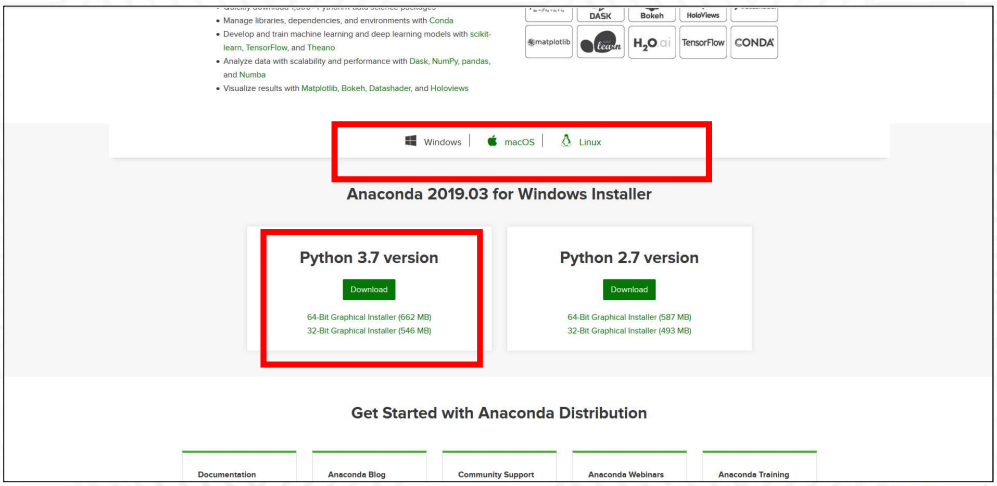
- Python 3 version of Anaconda
- Spyder as the IDE

- Spyder is bundled with Anaconda
- Anaconda is available at https://www.anaconda.com/distribution/

Installation instructions

- Install the proper version of Anaconda 3
  - Windows
  - MacOS
  - Linux
- Start Spyder
  - Accept defaults
  - …But you may want to specify a different directory for storing your work

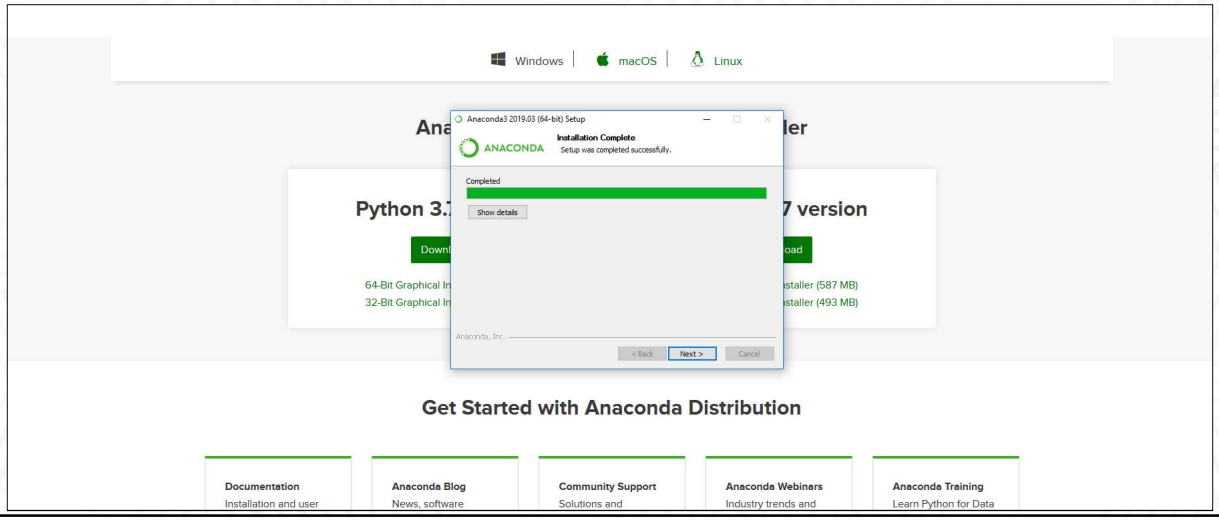
Select the proper version of Anaconda 3



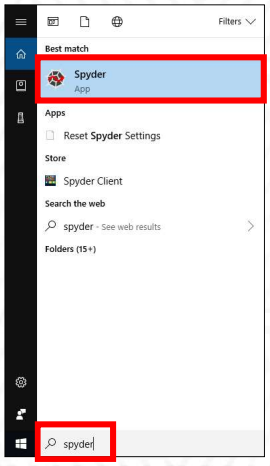
ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

Download Anaconda 3 and accept defaults



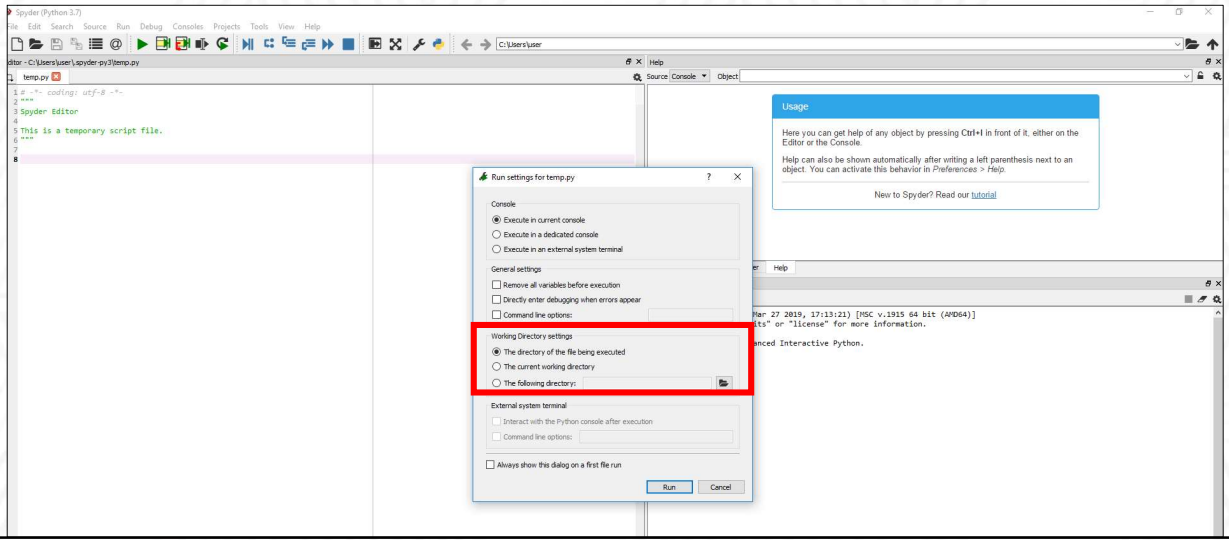
Find and start Spyder



ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

Accept the Spyder defaults



An example Python program

- Python files usually end with the suffix .py
- The program code below, called small.py, contains source code

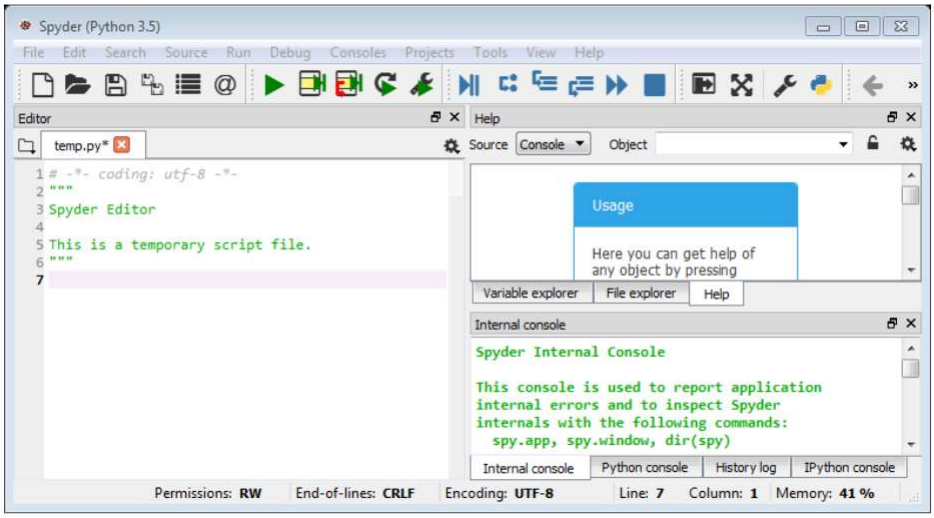
Filename: small.py

```
print("hello")
```

Output

```
hello
```


Spyder



Assignment: Spyder

- Start Spyder
- Work with default initial file
- Create a very simple program: `print("hello")`
- Save the .py file
- Run the program
- Verify the output

Variables

- Store a piece of information for later use (calculations, logic, input, output, etc.)
- Format

```
<name of variable> = <Information to be stored in the variable>
```

- Examples
  - Integer (e.g., `num1 = 10`)
  - Floating point (e.g., `num2 = 10.25`)
  - String (e.g., `name = "James"`)

Variable naming rules

- Variable names are case sensitive and cannot start with a number.
- They can contain letters, numbers, and underscores.

```
bob     Bob     _bob     _2_bob_     bob_2     BoB
```

- Examples

```
bob2 = 25
bob_2 = "Twenty-five"
```


University of Saskatchewan | SSRL

Simple data types

- Numbers
  - Integer
  - Floating-point
- Strings are a sequence of text characters
- Booleans are `False` or `True`

ssrl.usask.ca

University of Saskatchewan | SSRL

Numeric data types

- Integers

```
2, 25, -1024
```

- Floating-point

```
3.456, -5.25, 10.15
```


Arithmetic operators

Operator	Description	Example
=	Assignment	num **=** 7
+	Addition	num = 2 **+** 2
-	Subtraction	num = 6 **-** 4
*	Multiplication	num = 5 ***** 4
/	Division	num = 25 **/** 5
%	Modulo	num = 11 **%** 3
**	Exponent	num = 9 ****** 2

Order of operation

```
a = 5
b = 10
c = 32
d = 44

a+b*c-d/a+c
```

How should this be evaluated?

Order of operation

- Level of precedence: top to bottom
- If there are multiple operations on the same level, then precedence is evaluated in the order that they appear in the equation

()	Parentheses (inner before outer)
**	Exponent
*, /, %	Multiplication, division, modulo
+, -	Addition, subtraction

Order of operation and style

- It is good style to explicitly bracket your operations.

```
x = (a * b) + (c / d)
```

- It makes it easier to read complex formulas, and it is a good habit to adopt

Order of operation

```
a = 5
b = 10
c = 32
d = 44

a+b*c-d/a+c
```

How should this be evaluated?

```
(a+b)*(c-d)/(a+c)
```

Using parentheses makes the order clear.

Comments

- Comments begin with **#**, and then the rest of the line is ignored
- Examples

```
result = 1     #holds calculation result

#Pythagorean Theorem
side_a = sqrt((side_c * side_c) - (side_b * side_b))
```


Strings

- A sequence of text characters in a program.
  - Strings start and end with quotation mark " or apostrophe ' characters.
- Examples

```
"hello"
'This is a string'
"This is a string. It can be very long!"
```

Strings

- A string can include special characters (tab, newline, etc.)
- These are preceded by a backslash
  - `\t` tab character
  - `\n` new line character
  - `\"` quotation mark character
  - `\\` backslash character
  - Example

```
"Hello\tthere\nHow are you?"
```

```
Hello   there
How are you?
```


Displaying string output

- The **print** function is invoked, and the string is nested in parentheses
- Don't mix and match different types of quotation marks

- Examples

```
print ("foo")
foo
print ('bar')
bar
```

String concatenation

- Two or more strings can be joined with , or +
- , inserts a space, + does not

```
print( "hello"+"world" )
helloworld

print( "hello","world" )
hello world

age = 45
print ("You have", 65 - age, "years until retirement")
You have 20 years until retirement
```


String indexes

- Characters in a string are numbered with *indexes* starting at 0
- Accessing an individual character of a string:
  <variableName> [index]

- Example

```
name = "Nathalie"
```

index	0	1	2	3	4	5	6	7
character	N	a	t	h	a	l	i	e

```
print (name, "starts with", name[0])
Nathalie starts with N
```

String slicing

- Slicing can be used to select a substring
- The syntax is [start: end]
- The end index excludes the final character

```
print("hello"[1:4] )
ell

a = "purple flowers"
b = a[7:14]
print (b)
flowers
```


Strings

- Concatenation

```
print( "hello"+"world" )
helloworld

print( "hello","world" )
hello world
```

- Indexing

```
print( "hello"[0] )
h
```

- Slicing

```
print( "hello"[1:4] )
ell
```

String properties

- len(string) - number of characters in a string
- string.lower() - lowercase version of a string
- string.upper() - uppercase version of a string

- Example

```
name = "Sir Isaac Newton"
length = len(name)
big_name = name.upper()
print (big_name, "has", length, "characters")

SIR ISAAC NEWTON has 16 characters
```


Compound data types: List

- A container that holds a number of other objects, in a **given order**
- Defined using **square brackets**

```
a = [1, 2, 3, 4, 5]
print (a)
[1, 2, 3, 4, 5]
```

List indexing and slicing

- Similar to the syntax used for strings

```
a = [1, 2, 3, 4, 5]
print (len(a))
5

print (a[1])
2

print(a[2:4])
[3, 4]
```


List modification

- Items can be inserted and removed from lists
- The insert() function takes two arguments. The first is the index, and the second is the item to be inserted into the list.
- The pop() function removes the item at the specified index.

```
a = [1, 2, 3, 4, 5]
a.insert(0, 5.5)
print (a)
[5.5, 1, 2, 3, 4, 5]

a.pop(0)
print (a)
[1, 2, 3, 4, 5]
```

Lists

- Indexing

```
a = [4,3,2,1,0]
print ( a[1] )
3
```

- Insert

```
a.insert(0, 5.5)
print (a)
[5.5,4,3,2,1,0]
```

- Pop

```
a.pop(0)
print (a)
[4,3,2,1,0]
```

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

Lists

- Concatenation

```
a = a + a
print(a)
[4, 3, 2, 1, 0, 4, 3, 2, 1, 0]
```

- Slicing

```
a = a[0:5]
print(a)
[4, 3, 2, 1, 0]
```

- Length

```
print ( len(a) )
5
```

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

Data type wrap up

- Integers: 2323, 3234
- Floating Point: 32.3
- String: “Hello”
- Lists: l = [1,2,3]

Control flow

- Logic
- Whitespace
- If / then /else
- While loops
- For loops

Whitespace

- Whitespace is meaningful in Python
- Use consistent indentation to mark blocks of code
  - First line with *less* indentation is outside of the block
  - First line with *more* indentation starts a nested block
- Colons (:) start of a new block

```
b = 6
if b > 5:
    print("Greater than 5")
Greater than 5
```


Logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False

If, if/else, if/elif/else

- **If** statements are used to conditionally execute a block of code.
- The code is executed when the **if** statement evaluates to **TRUE**
- The **elif** (else if) keyword is evaluated if the previous conditions were not true
- The **else** keyword catches anything which isn't caught by the preceding conditions

If, if/else, if/elif/else (1)

```
a = 33
b = 200

if b > a:
   print("b is greater than a")
b is greater than a
```

If, if/else, if/elif/else (2)

```
a = 33
b = 33
if b > a:
   print("b is greater than a")
elif a == b:
   print("a and b are equal")
a and b are equal
```


If, if/else, if/elif/else (3)

```
a = 15
if a == 0:
   print ("zero!")
elif a < 0:
   print ("negative!")
else:
   print ("positive!")
positive!
```

Assignment

- Create a program that prompts the user for a numeric test grade
- The numeric grade should be used to determine if a student has earned an A, B, C, D or F in the class, using the rules on this table :

A	>= 90%
B	80.00-89.99%
C	70.00-79.99%
D	60.00-69.99%
F	<60.00

- Use a print() statement to output the letter grade

Assignment

- Copy and paste this code into a new Spyder window
- The code is available at: **https://github.com/pinelle**

```
score = input("What is your exam score? ")
score = float(score)
if (score >= 90) :
    print("A")
elif (score >= 80 and score < 90):
    print("B")
```

Common error messages

```
Python
def some_function()
    msg = "hello, world!"
    print(msg)
     return msg
```

```
Error
  File "<ipython-input-3-6bb841ea1423>", line 1
    def some_function()
                       ^
SyntaxError: invalid syntax
```

```
Python
def some_function():
    msg = "hello, world!"
    print(msg)
     return msg
```

```
Error
  File "<ipython-input-4-ae290e7659cb>", line 4
    return msg
    ^
IndentationError: unexpected indent
```


While loop

- Execute a set of statements as long as a condition is true.
- Requires a relevant test variables to be ready.
  - They are initially set outside of the loop
  - They are often counters.
- The **break** statement can be used to exit the loop (and the for loop) even if the while condition is true

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

While loop (1)

```
a = 10
while a > 0:
    print (a)
    a = a - 1
```

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


While loop (2)

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i = i + 1
```

```
1
2
3
```

For loop

- Used for iterating over a sequence (e.g. a list, a string, etc.).
- Executes a set of statements, once for each item
- Does not require an indexing variable to be set beforehand
- The range () function can be used to loop through a set of code a specified number of times
  - It returns a sequence of numbers, starting with 0 by default
  - For example, range(6) is not the value of 0 to 6, but the values 0 to 5

For loop (1)

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
   print(x)
```

```
apple
banana
cherry
```

For loop (2)

```
for a in range(10):
    print (a)
```

```
0
1
2
3
4
5
6
7
8
9
```


For loop (3)

```
a = [3, 1, 4, 1, 5, 9]
for i in range(len(a)):
    print (a[i])
```

```
3
1
4
1
5
9
```

Functions

- A function is a block of code that only runs when it is called
- Some functions accept data (known as parameters)
- Functions can return data
- In Python a function is defined using the **def** keyword:
- To call a function, use the function name followed by parenthesis

```
def my_function():
    print("Hello from a function")

my_function()
Hello from a function
```


Defining a function (1)

```
def foo(x):
    y = 10 * x + 2
    return y
```

- All variables are local and are not visible outside of the function

Executing a function (1)

```
def foo(x):
    y = 10 * x + 2
    return y

print (foo(10))
102
```


Defining a function (2)

```
def  add_numbers (arg1, arg2, arg3):
     sum = arg1 + arg2 + arg3
     return sum
```

Executing a function (2)

```
def  add_numbers (arg1, arg2, arg3):
     sum = arg1 + arg2 + arg3
     return sum

print ( add_numbers(5, 20, 25) )
50
```


Assignment

- Write a function that calculates whether or not a given year is a leap year and has 366 days instead of 365.
- Use this algorithm:
  - A year will be a leap year if it is divisible by 4 but not by 100
  - If a year is divisible by 4 and by 100, it is not a leap year unless it is also divisible by 400
- The function should print :"The year X is a leap year." or "The year X is not a leap year."
- It should accept one argument: a non-negative number representing a year to evaluate

Assignment

- Copy and paste this code into a new Spyder window
- The code is available at: **https://github.com/pinelle**

```
def  leapYear (year):
    #determine if this is a leap year
    #if is leap year, print ("The year",year,"is a leap year.")
    #otherwise, print ("The year",year,"is not a leap year.")
    if year % 4 == 0 and year % 100 != 0:
        ...

leapYear(1996) #true
leapYear(2000) #true
leapYear(2002) #false
leapYear(1600) #true
```


Modules

- Access other code by importing modules

```
import math
print (math.sqrt(2.0))
```

- or:

```
from math import sqrt
print (sqrt(2.0))
```

Modules

- Import a module and assign a variable
- This often makes it easier to work with multiple modules

```
import numpy as np
import pandas as pd
import math as m

print("The value of pi is", m.pi)
a = np.array([1, 2, 3])
s = pd.Series([1, 3, 5, np.nan, 6, 8])
```


Introduction to Numpy

- NumPy stands for “Numeric Python”
- Typed multi-dimensional arrays (matrices)
  - **Fast** numerical computations (matrix math)
  - High-level math functions
- Numpy arrays are used in Python machine learning toolkits and in scientific and engineering applications

The Numpy Ndarray

- The most important object defined in NumPy is an N-dimensional array type called ndarray.
- It is a collection of items of the same type, and they can be accessed using a zero-based index.

7
20
3
2
16

$$\begin{bmatrix} p_1 \\ p_2 \\ p_n \end{bmatrix}$$

5.1	3.5	1.4	0.2
4.9	3	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.3	0.2
5	3.6	1.4	0.2

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

UNIVERSITY OF SASKATCHEWAN | SSRL

NumPy array indexing, slicing

```
x[0,0]      # top-left element
x[0,-1]     # first row, last column
x[0,:]      # first row (many entries)
x[:,0]      # first column (many entries)
```

Notes

- Arrays can be negatively indexed, starting from the end
- Multi-dimensional indexes are comma-separated (i.e., a tuple)

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

NDArrays: Creation and conversion

```
import numpy as np

# Create a Python List
my_list = [1, 3, 5, 7, 9]
print ("Python list : ", my_list,)
Python list :  [1, 3, 5, 7, 9]

# Create a new Numpy NDArray using the List
my_ndarray = np.array(my_list)
print("Numpy NDArray : ", my_ndarray)
Numpy NDArray :  [1 3 5 7 9]
```


NDArrays: Indexing and slicing

```
#create a ndarrray with arange
a = np.arange(10)
print(a)
[0 1 2 3 4 5 6 7 8 9]

#slice a single item
print( a[5] )
5

#slice items starting from an index
print( a[2:] )
[2 3 4 5 6 7 8 9]
```

NDArrays: Indexing and slicing

```
[0 1 2 3 4 5 6 7 8 9]
#slice items between indexes
print (a[2:5])
[2 3 4]

a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a)
[[1 2 3]
 [3 4 5]
 [4 5 6]]

#slice rows starting from index
print( a[1:] )
[[3 4 5]
 [4 5 6]]
```


NDArrays: Indexing and slicing

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]

#slice rows up to an index
print( a[:1])
[[1 2 3]]

#slice the second column
print( a[:,1] )
[2 4 5]
```

NDArrays: Shape

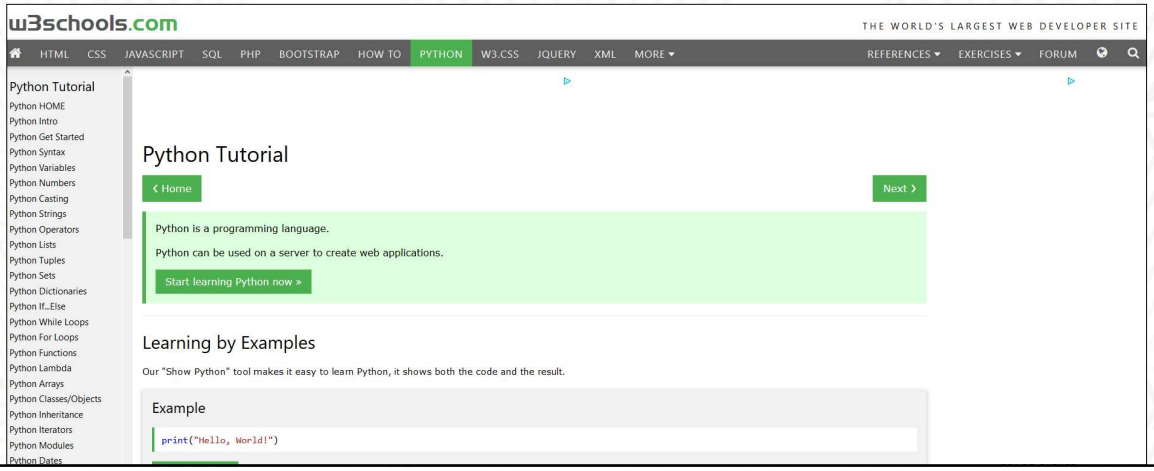
```
a = np.array([[1,2,3],[4,5,6]])
print(a)
[[1 2 3]
 [4 5 6]]
print (a.shape)
(2, 3)

#this resizes the ndarray
a.shape = (3,2)
print(a)
[[1 2]
 [3 4]
 [5 6]]
```

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

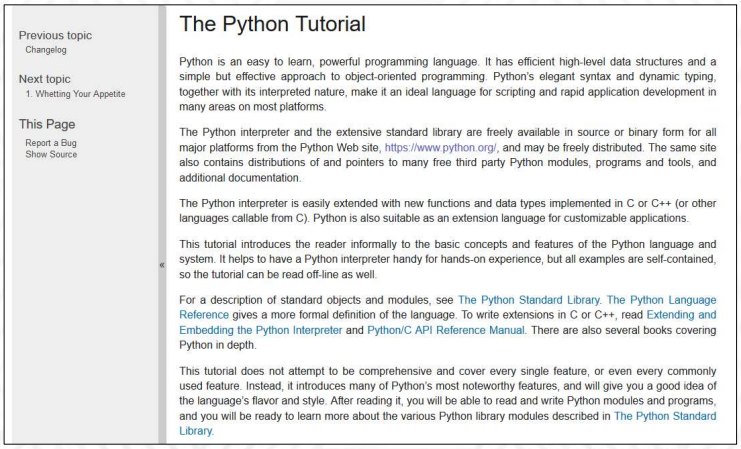
W3 Schools

- https://www.w3schools.com/python/



Python Tutorial

- https://docs.python.org/3/tutorial/



Previous topic
Changelog

Next topic
1. Whetting Your Appetite

This Page
Report a Bug
Show Source

The Python Tutorial

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in The Python Standard Library.

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL