

Introductory Programming in Python

Day 1



Pre course survey

When survey is active, respond at **PollEv.com/kheeweiseow794**

0 surveys done

↻ 0 surveys underway

Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app

To show this poll

1

Install the app from
pollev.com/app

2

Start the presentation

Still not working? Get help at pollev.com/app/help
or

[Open poll in your web browser](https://pollev.com)

To show this poll

1

Install the app from
pollev.com/app

2

Start the presentation

Still not working? Get help at pollev.com/app/help
or

[Open poll in your web browser](https://pollev.com)

To show this poll

1

Install the app from
pollev.com/app

2

Start the presentation

Still not working? Get help at pollev.com/app/help
or

[Open poll in your web browser](https://pollev.com)



Introduction of trainer



Name
Seow Khee Wei

Email
seow_khee_wei@rp.edu.sg

Phone
98463112

Telegram
@kwseow



Bachelor

BEng major in computer
engineering
(Nanyang Technological
University)



Master

MBA
University of Melbourne



Post Graduate Certificate

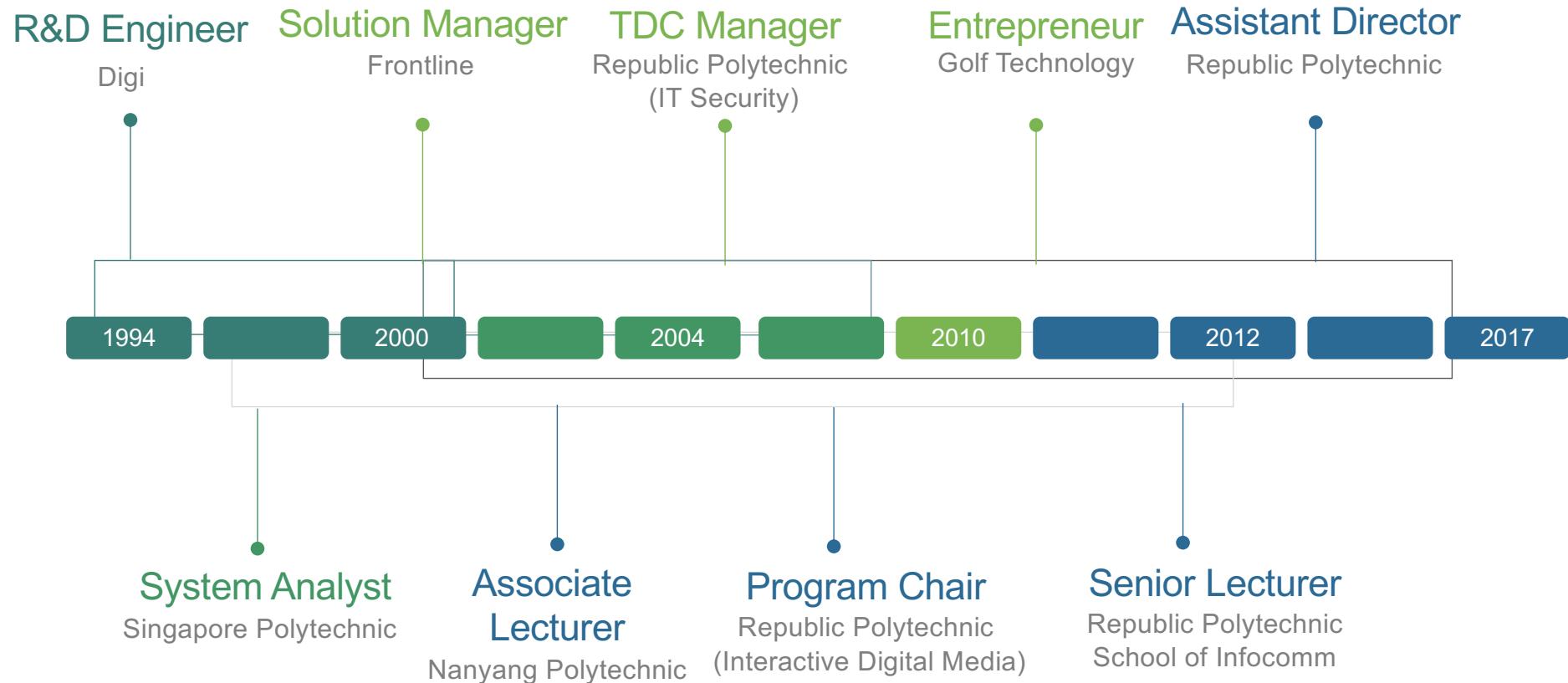
Network Engineering
(NTU)



Nano Degree

Artificial Intelligence

Introduction of trainer



Portfolio

Technology Development Centre

Cognitive Systems Technology Centre
Data Analytics Technology Centre
IoT Solutions Centre

Industry Joint-Lab

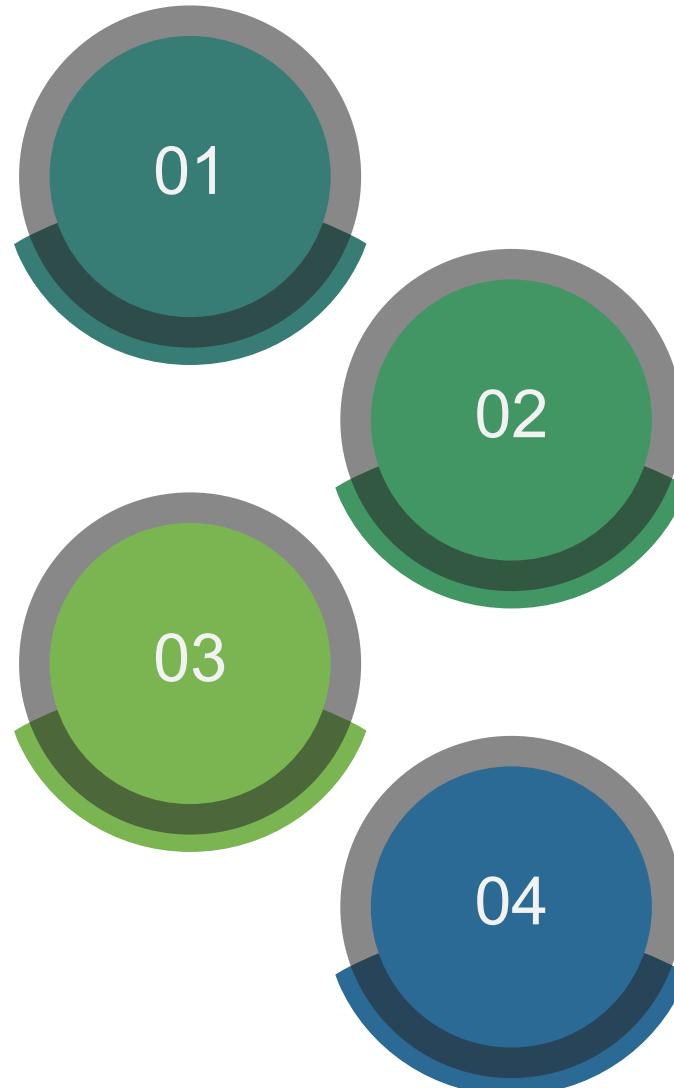
Microsoft, Samsung, Element14, Starhub, RSA,
Palo Alto Network, UofG, Secura, Ixia, Trend Micro

Industry Engagement

Student Internship, FYP
Staff Attachment, Consultancy Projects

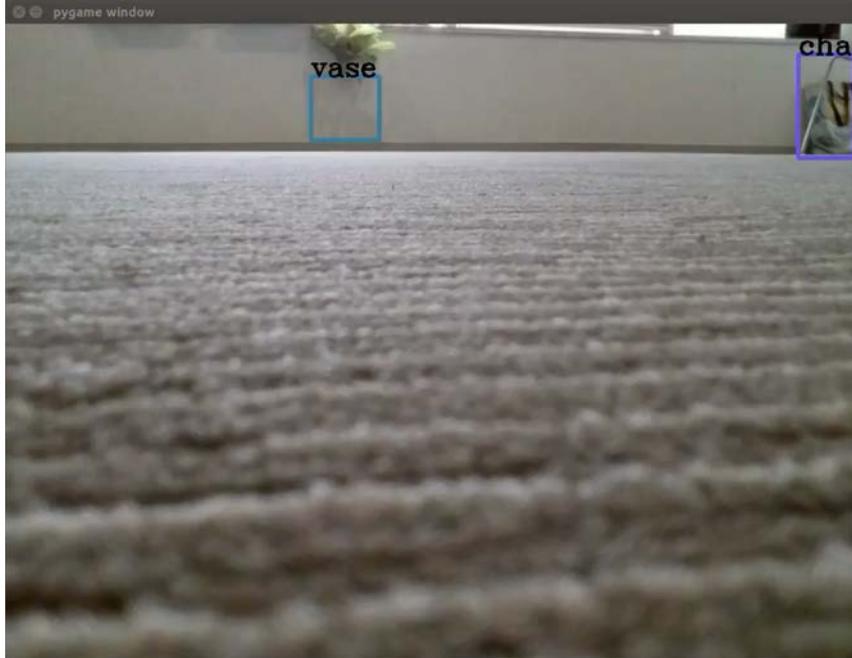
Training

Iphone Development, Programming, Artificial
Intelligence



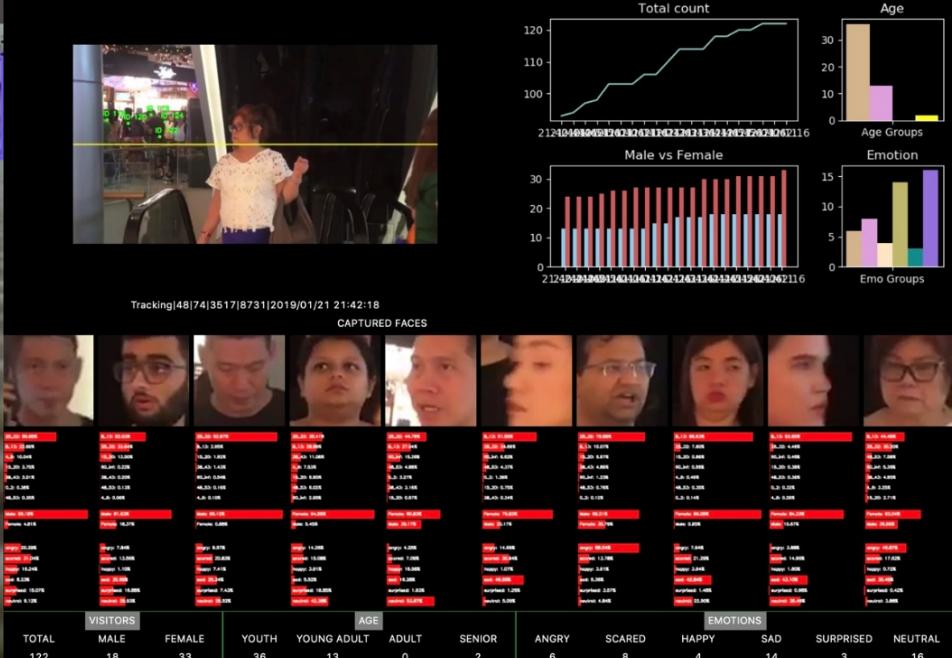
Pet Project

pygame window



Tracking|48|74|3517|8731|2019/01/21 21:42:18

CAPTURED FACES



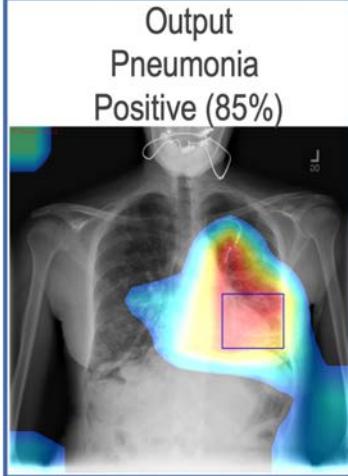
	VISITORS	AGE	EMOTIONS
TOTAL	122	0	14
MALE	18	6	SAD
FEMALE	33	8	SCARED
YOUTH	36	4	HAPPY
YOUNG ADULT	13	3	SURPRISED
ADULT	0	16	NEUTRAL
SENIOR	2		
ANGRY			
SCARED			
HAPPY			
SAD			
SURPRISED			
NEUTRAL			

Input Chest X-Ray Images

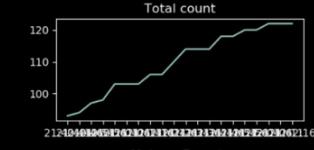


CheXNet
121 layers CNN

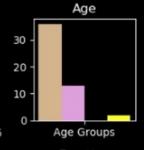
Output Pneumonia Positive (85%)



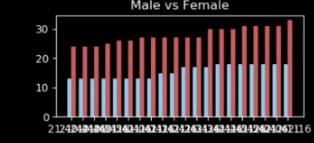
Total count



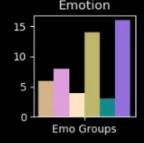
Age Groups



Male vs Female



Emotion



Emo Groups

Why are we here?



“Everybody in this country should learn how to program a computer... because it teaches you how to think.”

- Steve Jobs

About this workshop

- Learn about Python 3, a very versatile and useful language
- Discuss its advantages and disadvantages (also what to look out for)
- Improve your problem solving skills:
How to automate the most boring and repetitive stuff using Python
- Some tools and useful modules you can use to build your applications

Primary Focus

- This course is an introduction to Python Programming
- Designed for people seeking to improve their efficiency at work (and life!)
- This course merges two topics:
Learning Python + becoming a problem solver using code
- You will learn Python through solving problems:
Learn by doing!

Approach

- Coding coding coding (Steve Jobs)
- Learn as we need it (and how to be resourceful)
- Ask questions and for help when you get stuck.
- Don't be afraid to try new stuff (and make mistakes)
- How to use (aka read) the documentation

Prereqs and preparations

Before you attend this workshop, please make sure:

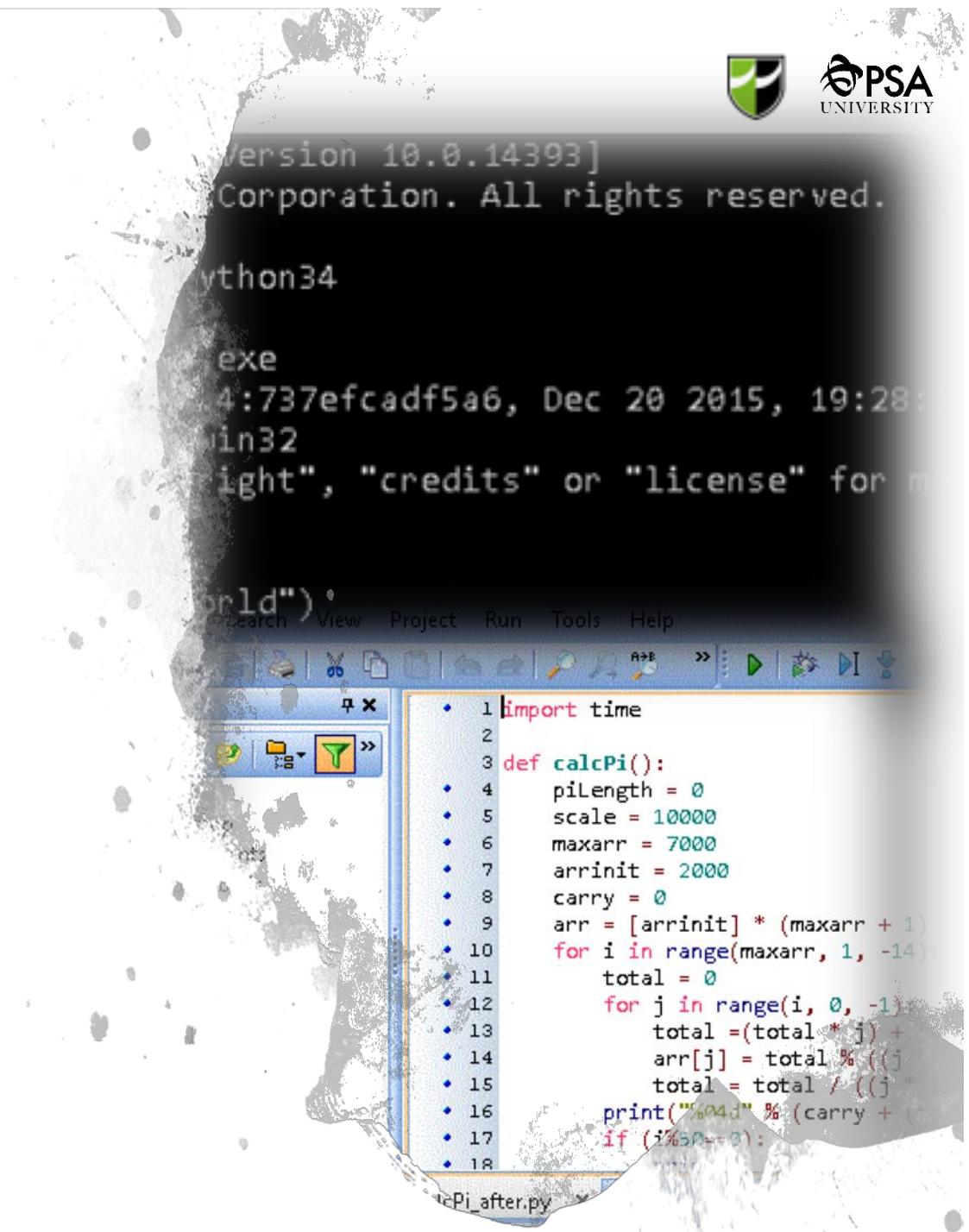
- Your laptop works



- You have installed Anaconda 5.3.0 for Python 3.7
- You installed a decent editor, We are using PyCharm Community Edition in this course
- You should (preferably) have some basic programming experience and be familiar with basic concepts such as variables, flow control and functions

Programme Day One

- A brief history of Python
- Setting up python environment
- Learn the basics:
 - Data types
 - Conversions
 - String operations
 - Functions
- String functions, formatting
- Find files by name, by extension, by size, by content and calculate the total size
- Graphical User Interface



Version 10.0.14393]
Corporation. All rights reserved.

python34
exe
4:737efcadf5a6, Dec 20 2015, 19:28:
in32
ight", "credits" or "license" for m

world")
Search View Project Run Tools Help

```
1 import time
2
3 def calcPi():
4     piLength = 0
5     scale = 10000
6     maxarr = 7000
7     arrinit = 2000
8     carry = 0
9     arr = [arrinit] * (maxarr + 1)
10    for i in range(maxarr, 1, -14):
11        total = 0
12        for j in range(i, 0, -1):
13            total = (total * j) +
14            arr[j] = total % ((j *
15            total = total / ((j *
16            print("%04d" % (carry +
17            if (i%50==0):
```

calcPi_after.py

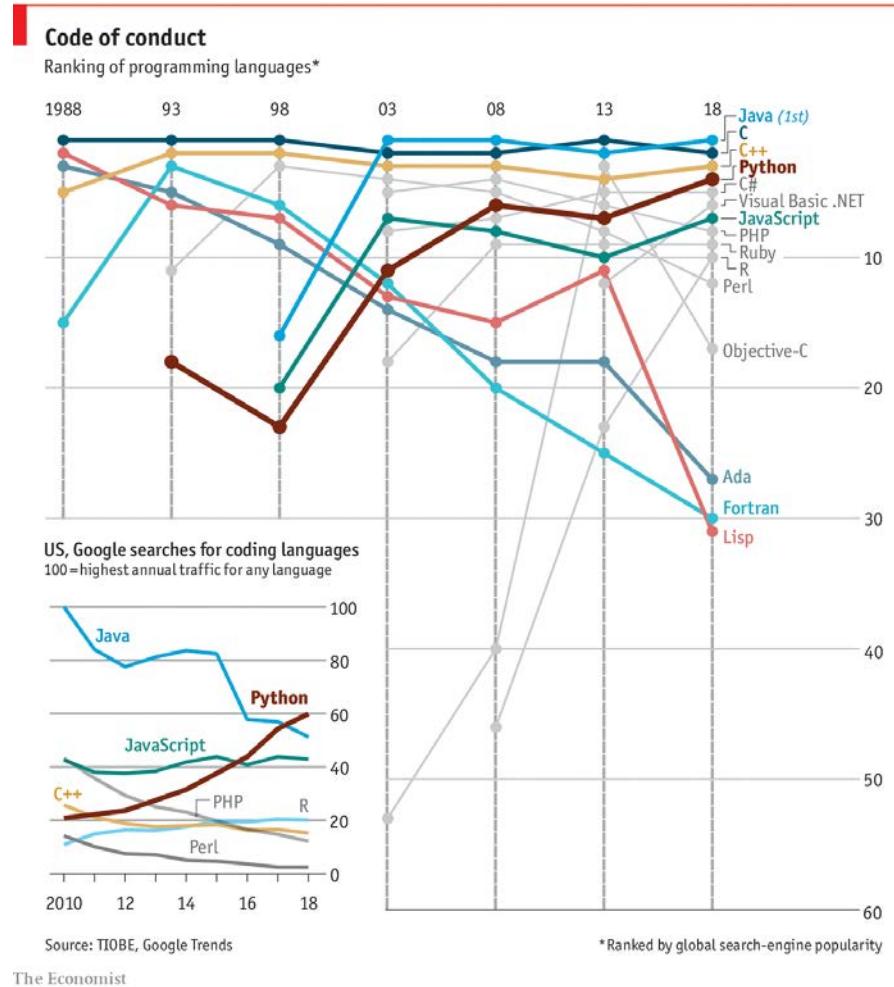
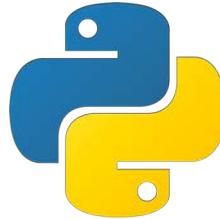
Programme Day Two

- Read and writing files
 - Copying, moving and deleting files and folders
 - Working with Excel
 - Processing CSV files
 - Generating PDF
 - Image processing
 - Data Visualization - Charting
 - Connecting to the Web
 - Sending emails
 - Telegram bot

Introduction to Python

What is Python?

- Interpreted
 - Interactive
 - Functional
 - Object-oriented
 - Programming language,
not just a scripting languageThe Python logo consists of two interlocking snakes, one blue and one yellow, forming a stylized 'P' shape.



Introduction to Python

- Allows modular programming
- Great emphasis on readability:
 - Codes are forced to be indented
 - But there's a practical drawback to this
- Easy to embed in and extend with other languages
- Easy to learn for beginners
- Completely FREE! Open source
- Copyrighted but use not restricted
- Interpreted
- Multiprardigm
- Multipurpose
- Cross-platform
- Dynamically typed
- Indentation aware
- Garbage collecting

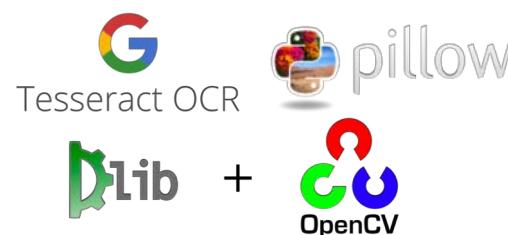
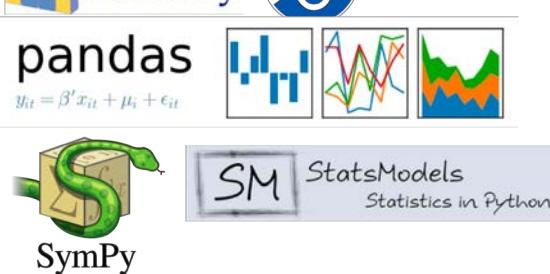
Introduction to Python

- Interfaces to:

- COM, DCOM, ODBC
- Most databases (MySQL, MSQL, Sybase, Oracle,...)
- Java (Jpython)
- Many GUI / GFX libraries
 - Platform-independent: Tk, wxWindows, GTK, Pyglet
 - Platform-specific: MFC, MacOS, X11



matplotlib



Introduction to Python

Python vs. Perl:

- Easier to learn
- More readable
- Fewer side effects
- Less Unix bias

Python vs. Tcl:

- Much faster
- Less need for C extensions
- Better Java integration

Python vs. Java:

- More concise code
- Dynamic typing
- Runs slower but development is fast
- No native-code compilation
- Can be integrated with Java using JPython

Nov 2018	Nov 2017	Change	Programming Language	Ratings	Change
1	1		Java	16.746%	+3.51%
2	2		C	14.396%	+5.10%
3	3		C++	8.282%	+2.94%
4	4		Python	7.683%	+3.20%
5	7	▲	Visual Basic .NET	6.490%	+3.58%
6	5	▼	C#	3.952%	+0.94%
7	6	▼	JavaScript	2.655%	-0.32%
8	8		PHP	2.376%	+0.48%
9	-	▲	SQL	1.844%	+1.84%
10	14	▲	Go	1.495%	-0.07%
11	19	▲	Objective-C	1.476%	+0.06%
12	20	▲	Swift	1.455%	+0.07%
13	9	▼	Delphi/Object Pascal	1.423%	-0.32%
14	11	▼	R	1.407%	-0.20%
15	10	▼	Assembly language	1.108%	-0.61%
16	13	▼	Ruby	1.091%	-0.50%
17	12	▼	MATLAB	1.030%	-0.57%
18	15	▼	Perl	1.001%	-0.56%
19	18	▼	PL/SQL	1.000%	-0.45%
20	17	▼	Visual Basic	0.854%	-0.63%

<https://www.tiobe.com/tiobe-index/>

Introduction to Python

Who uses Python?

01

Web Development. Google (in search spiders), Yahoo (in maps applications)

02

Games. Civilization 4 (game logic & AI), Battlefield 2 (score keeping and team balancing)

03

Graphics. Industrial Light & Magic (rendering), Blender 3D (extension Language)

04

Financial. ABN AMRO Bank (communicate trade information between systems).

05

Science. National Weather Centre, US (make maps, create forecasts, etc) NASA (Integrated Planning System)

06

Education. University of California, Irvine. University of New South Wales (Australia), NUS, SUTD, SMU, RP

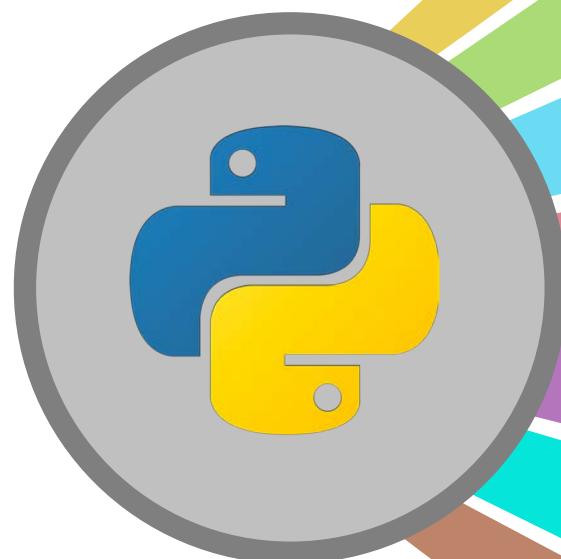
07

Business Software. Thawte Consulting, IBM, RealNetworks

More examples: <http://wiki.python.org/moin/OrganizationsUsingPython>

Introduction to Python

What is Python used for?



01 | Web and internet development (Django, Flask, Google App Engine)



02 | File storage (Dropbox)



03 | Scientific and Numeric computing (SciPy, NumPy)



04 | Data Science, Artificial Intelligence, Machine Learning (Keras, Scikit-learn, Pandas)



05 | Micro Controller (Micro-Python)



06 | Embedded in software packages (Maya, Blender)



07 | Scripting to perform simple (but mundane & repetitive) tasks

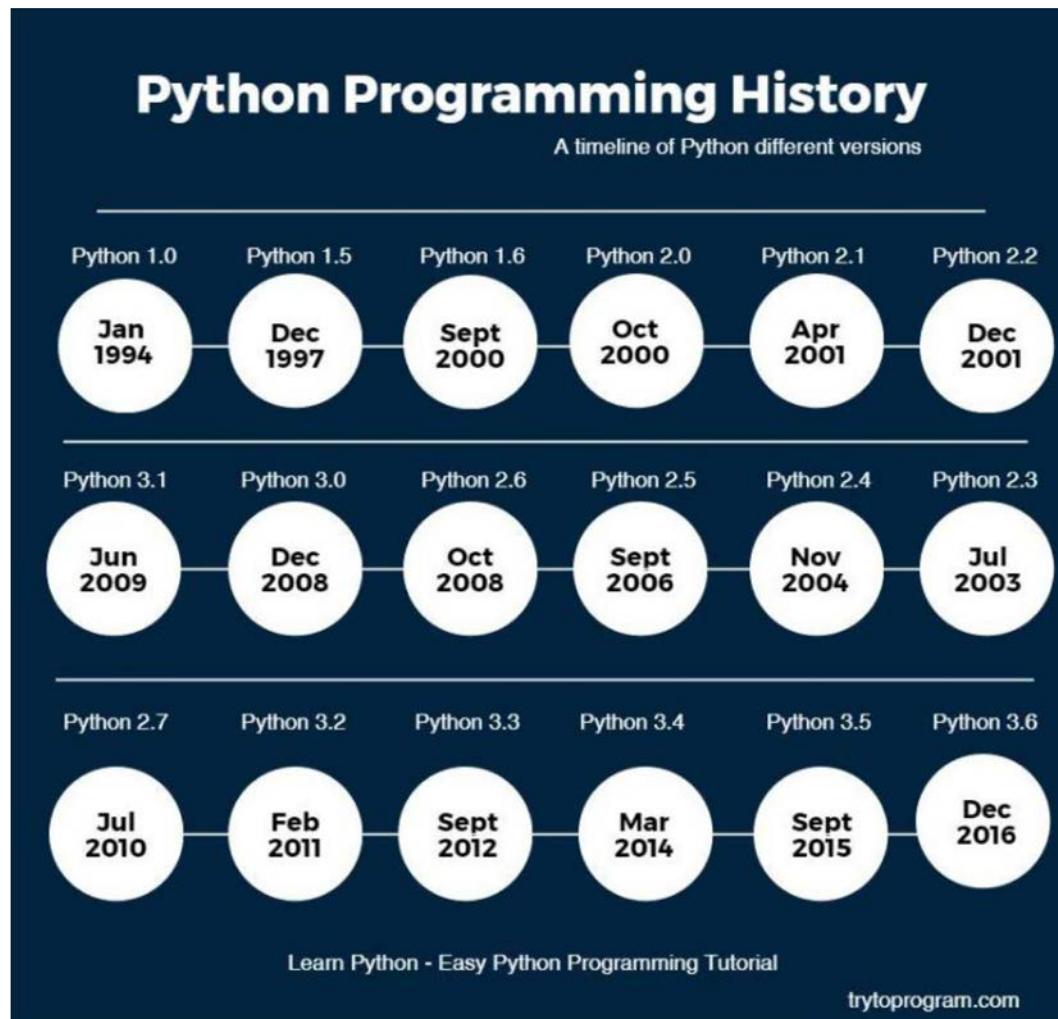


Introduction to Python

Why the name, Python?

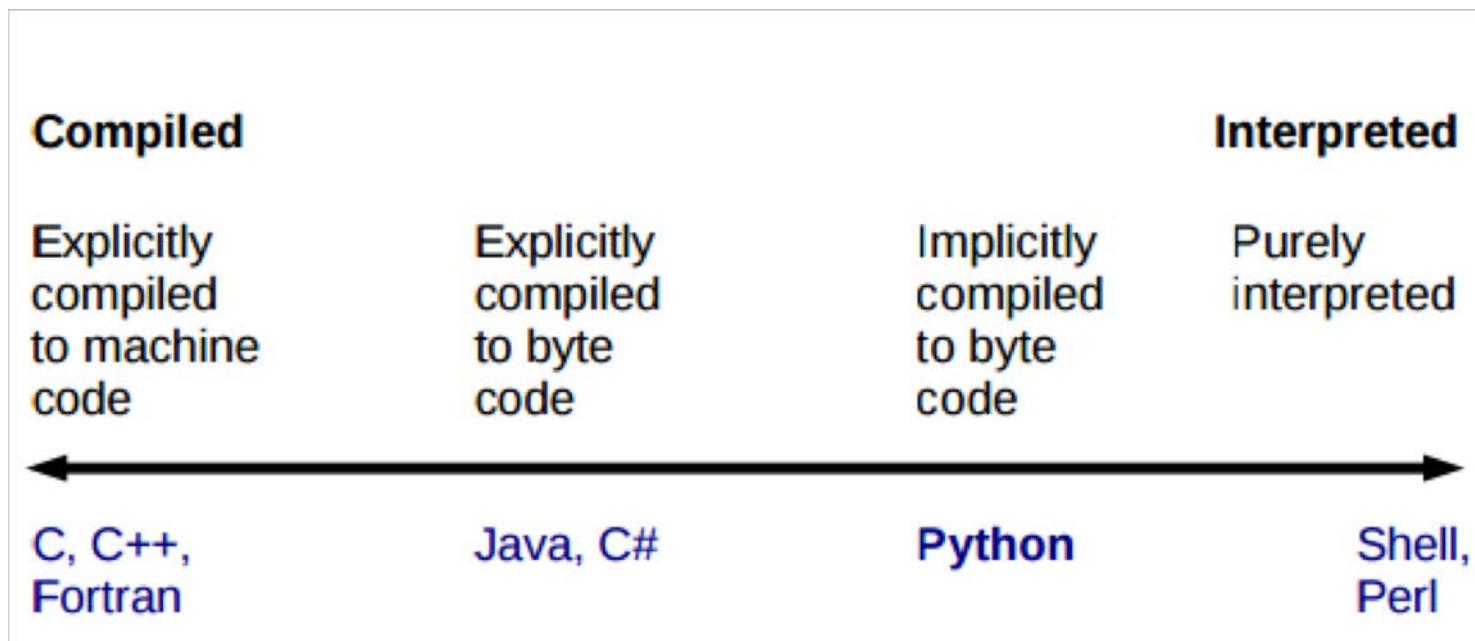
- Originally not a snake, but from the British comedy “**Monty Python’s Flying Circus**”. The snake logo came later.
- Invented in 1990 by Guido Van Rossum
- Initially intended to be a scripting language on Amoeba OS
- Python was influenced by ABC and Modula-3
- First public release was in 1991

Introduction to Python



Python has two versions currently: **2.7.14** and **3.7**

Where is Python on the Map



Python 2 vs. Python 3

- **Different syntax:** e.g. print statement, division
 - Python 2
 - ✓ `print "Hello World!"`
 - ✓ `x = 5 / 2` # x's value will be 2
 - Python 3
 - ✓ `print("Hello World!")` # brackets are compulsory now
 - ✓ `x = 5 / 2` # x's value will be 2.5
- **Which to learn?**
 - Many major frameworks and third-party modules have already migrated or are in the process of moving to Python 3
 - Python 2's EOL is in 2020, no Python 2.8
 - **The obvious pick: Python 3**

Getting started

Install and setup

- Get it @ <https://www.anaconda.com/download/>
- Use Python 3.7 for this course
- Set up virtual environments

How to write and run Python programs

- Use a text editor (notepad, vi, emacs, etc)
- Use an IDE (IDLE, Wing, PyCharm, etc)

Experiment with Jupyter Notebook

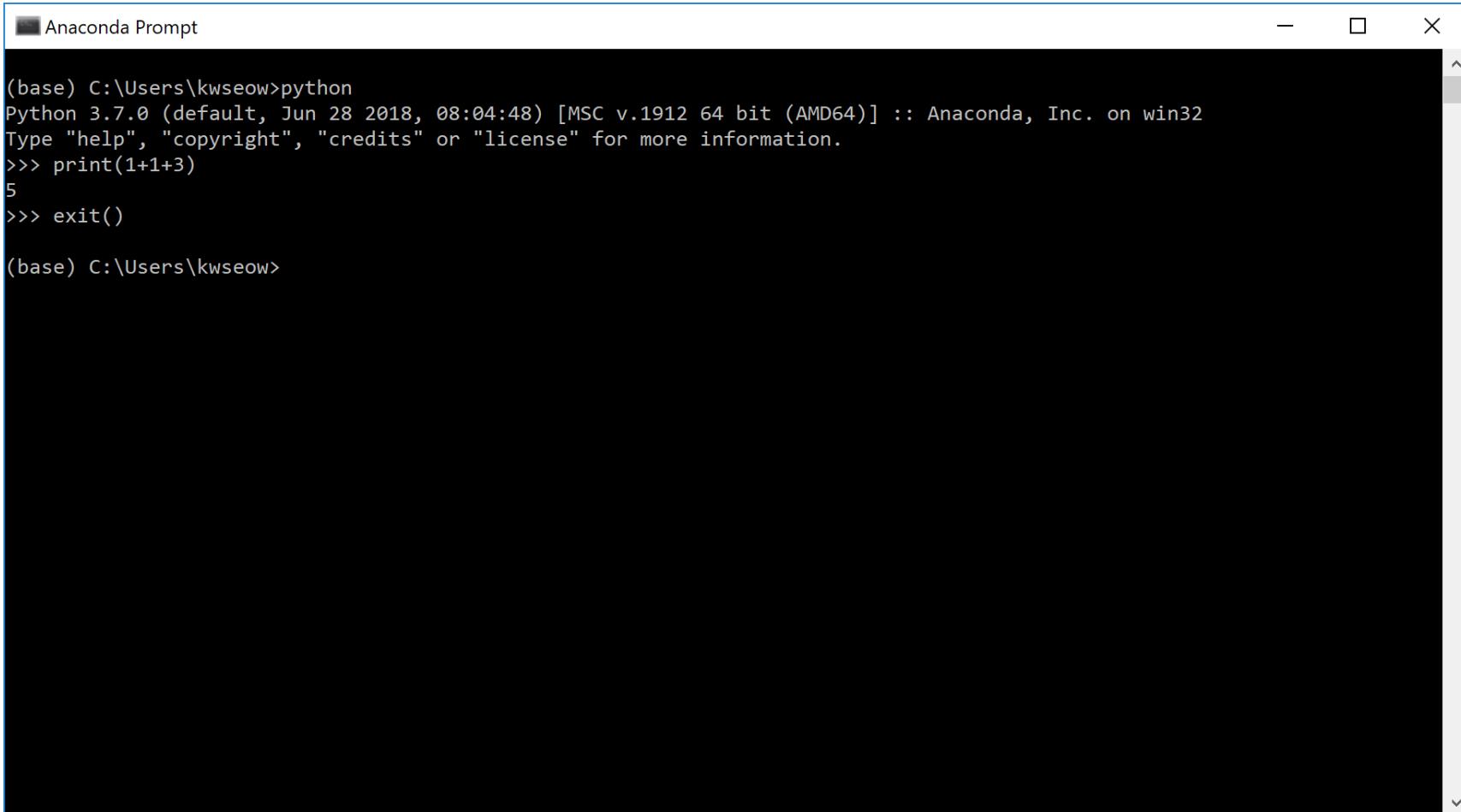
- Installed together with Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for data science and machine learning applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution is used by over 6 million users and includes more than 250 popular data-science packages suitable for Windows, Linux, and MacOS.

Running Python the basic way



First, Experience the Basic Way

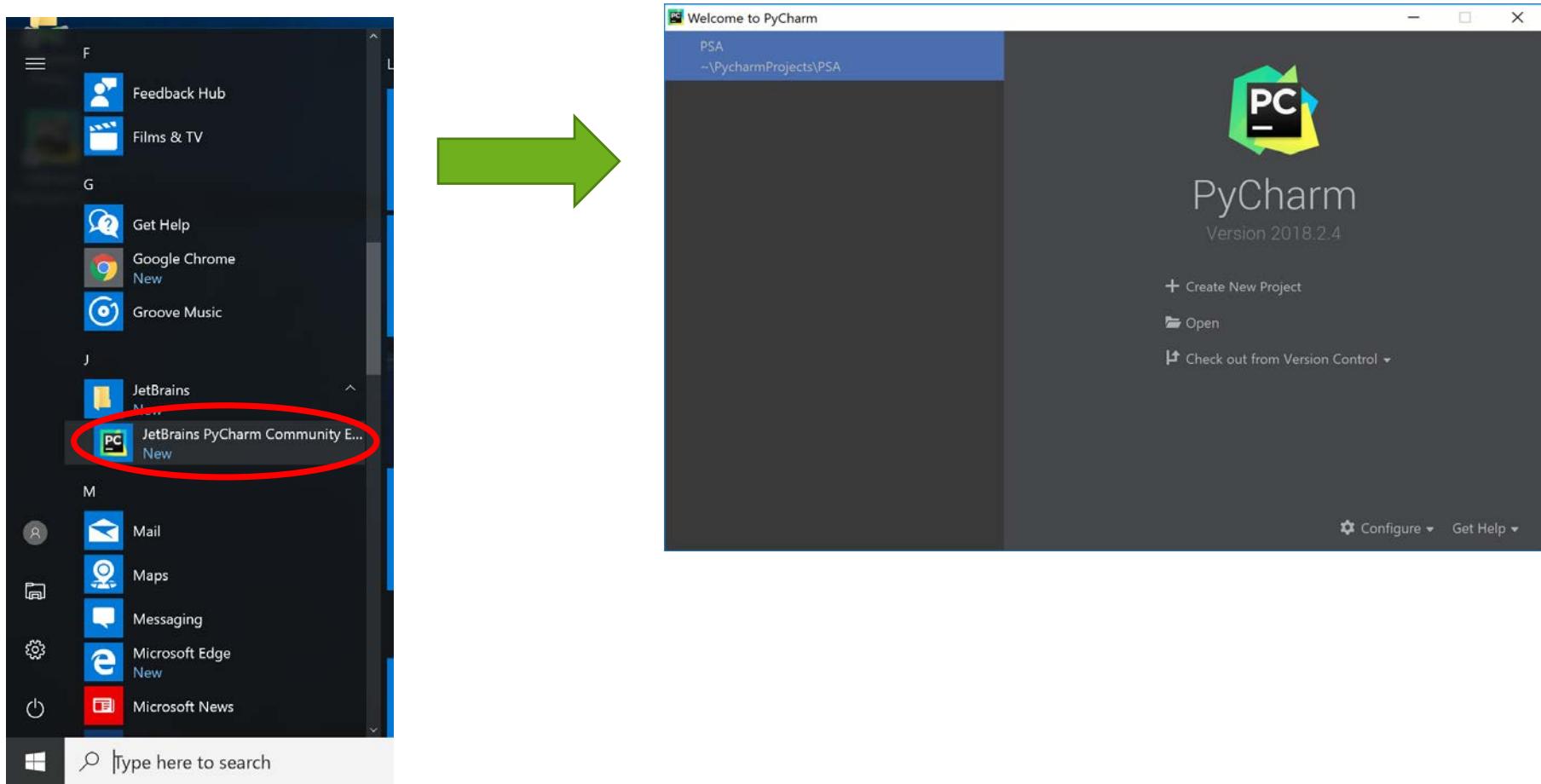


The screenshot shows a Windows-style terminal window titled "Anaconda Prompt". The command line interface displays the following Python session:

```
(base) C:\Users\kwseow>python
Python 3.7.0 (default, Jun 28 2018, 08:04:48) [MSC v.1912 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(1+1+3)
5
>>> exit()

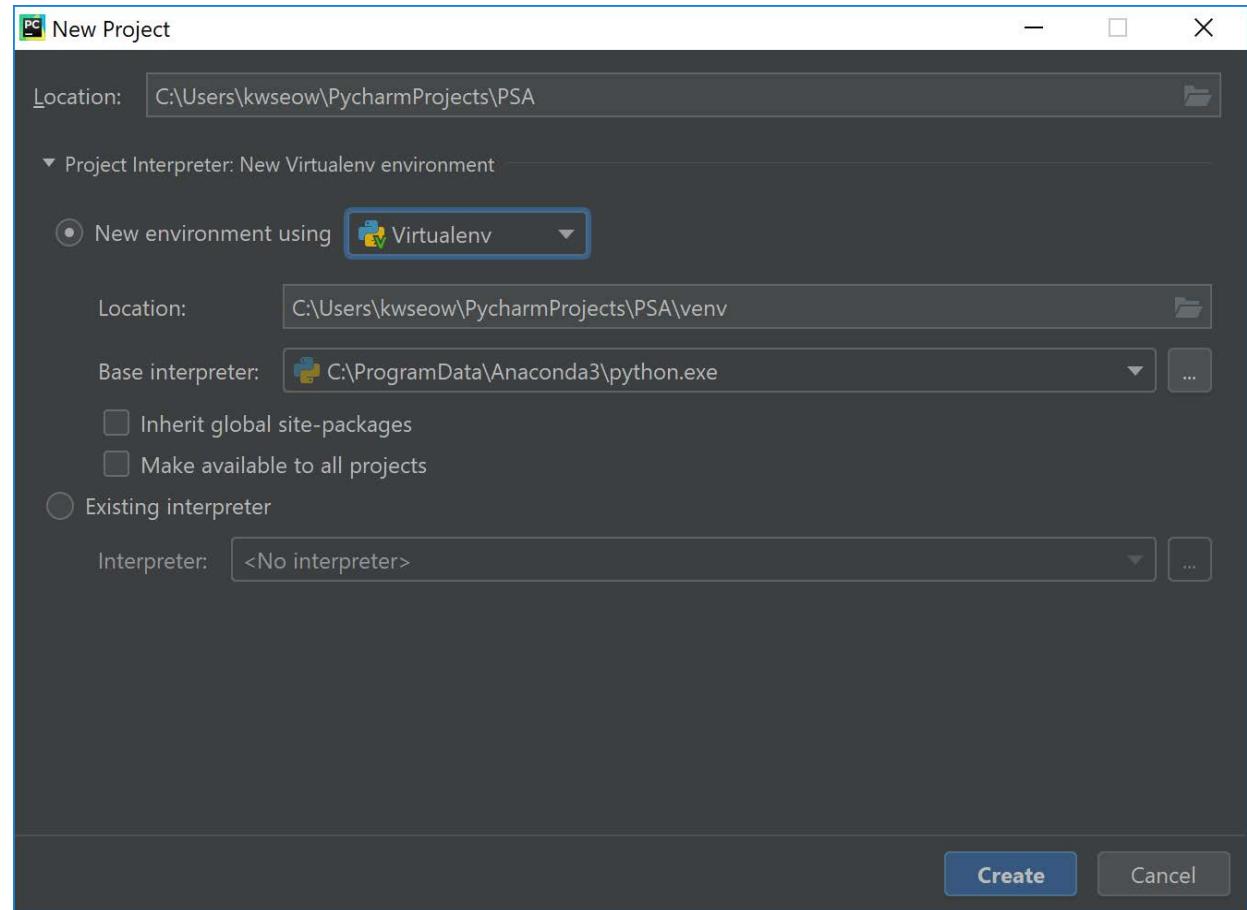
(base) C:\Users\kwseow>
```

Use an IDE, Run PyCharm



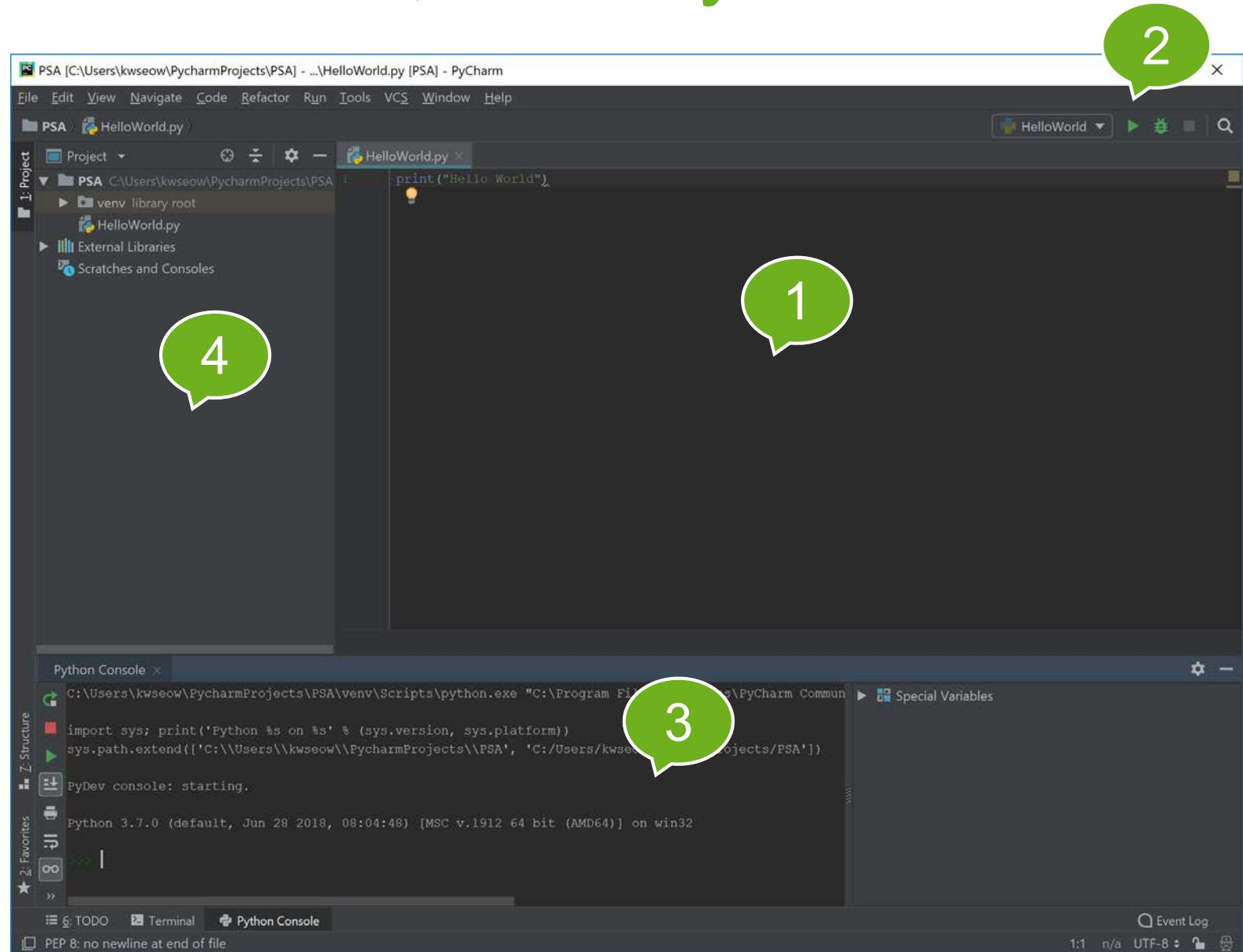
Use an IDE, Run PyCharm

- Enter the project info
- Use the system installed Python interpreter



Use an IDE, Run PyCharm

1. Editor
2. Run button
3. Output window / Console
4. Your files





DEFAULT KEYMAP

Editing

Ctrl + Space	Basic code completion (the name of any class, method or variable)
Ctrl + Alt + Space	Class name completion (the name of any project class independently of current imports)
Ctrl + Shift + Enter	Complete statement
Ctrl + P	Parameter info (within method call arguments)
Ctrl + Q	Quick documentation lookup
Shift + F1	External Doc
Ctrl + mouse over code	Brief Info
Ctrl + F1	Show descriptions of error or warning at caret
Alt + Insert	Generate code...
Ctrl + O	Override methods
Ctrl + Alt + T	Surround with...
Ctrl + /	Comment/uncomment with line comment
Ctrl + Shift + /	Comment/uncomment with block comment
Ctrl + W	Select successively increasing code blocks
Ctrl + Shift + W	Decrease current selection to previous state
Ctrl + Shift +]	Select till code block end
Ctrl + Shift + [Select till code block start
Alt + Enter	Show intention actions and quick-fixes
Ctrl + Alt + L	Reformat code
Ctrl + Alt + O	Optimize imports
Ctrl + Alt + I	Auto-indent line(s)
Tab	Indent selected lines
Shift + Tab	Unindent selected lines
Ctrl + X, Shift + Delete	Cut current line or selected block to clipboard
Ctrl + C, Ctrl + Insert	Copy current line or selected block to clipboard
Ctrl + V, Shift + Insert	Paste from clipboard
Ctrl + Shift + V	Paste from recent buffers...
Ctrl + D	Duplicate current line or selected block
Ctrl + Y	Delete line at caret
Ctrl + Shift + J	Smart line join
Ctrl + Enter	Smart line split
Shift + Enter	Start new line
Ctrl + Shift + U	Toggle case for word at caret or selected block
Ctrl + Delete	Delete to word end
Ctrl + Backspace	Delete to word start
Ctrl + NumPad+	Expand code block
Ctrl + NumPad-	Collapse code block
Ctrl + Shift + NumPad+	Expand all
Ctrl + Shift + NumPad-	Collapse all
Ctrl + F4	Close active editor tab

Running

Alt + Shift + F10	Select configuration and run
Alt + Shift + F9	Select configuration and debug
Shift + F10	Run
Shift + F9	Debug
Ctrl + Shift + F10	Run context configuration from editor
Ctrl + Alt + R	Run manage.py task

Debugging

F8 / F7	Step over/into
Shift + F8	Step out
Alt + F9	Run to cursor
Alt + F8	Evaluate expression
Ctrl + Alt + F8	Quick evaluate expression
F9	Resume program
Ctrl + F8	Toggle breakpoint
Ctrl + Shift + F8	View breakpoints

Navigation

Ctrl + N	Go to class
Ctrl + Shift + N	Go to file
Ctrl + Alt + Shift + N	Go to symbol
Alt + Right	Go to next editor tab
Alt + Left	Go to previous editor tab
F12	Go back to previous tool window
Esc	Go to editor (from tool window)
Shift + Esc	Hide active or last active window
Ctrl + Shift + F4	Close active run/messages/find... tab
Ctrl + G	Go to line
Ctrl + E	Recent files popup
Ctrl + Alt + Right	Navigate forward
Ctrl + Alt + Left	Navigate back
Ctrl + Shift + Backspace	Navigate to last edit location
Alt + F1	Select current file or symbol in any view
Ctrl + B , Ctrl + Click	Go to declaration
Ctrl + Alt + B	Go to implementation(s)
Ctrl + Shift + I	Open quick definition lookup
Ctrl + Shift + B	Go to type declaration
Ctrl + U	Go to super-method/super-class
Alt + Up / Down	Go to previous/next method
Ctrl + J / [Move to code block end/start
Ctrl + F12	File structure popup
Ctrl + H	Type hierarchy
Ctrl + Shift + H	Method hierarchy
Ctrl + Alt + H	Call hierarchy
F2 / Shift + F2	Next/previous highlighted error
F4	Edit source
Ctrl + Enter	View source
Alt + Home	Show navigation bar
F11	Toggle bookmark
Ctrl + Shift + F11	Toggle bookmark with mnemonic
Ctrl + #[0-9]	Go to numbered bookmark
Shift + F11	Show bookmarks

Search/Replace

Ctrl + F / Ctrl + R	Find/Replace
F3 / Shift + F3	Find next/previous
Ctrl + Shift + F	Find in path
Ctrl + Shift + R	Replace in path

Usage Search

Alt + F7 / Ctrl + F7	Find usages / Find usages in file
Ctrl + Shift + F7	Highlight usages in file
Ctrl + Alt + F7	Show usages

Refactoring

F5 / F6	Copy / Move
Alt + Delete	Safe Delete
Shift + F6	Rename
Ctrl + F6	Change Signature
Ctrl + Alt + N	Inline
Ctrl + Alt + M	Extract Method
Ctrl + Alt + V	Extract Variable
Ctrl + Alt + F	Extract Field
Ctrl + Alt + C	Extract Constant
Ctrl + Alt + P	Extract Parameter

VCS/Local History

Ctrl + K	Commit project to VCS
Ctrl + T	Update project from VCS
Alt + Shift + C	View recent changes
Alt + BackQuote (`)	'VCS' quick popup

Live Templates

Ctrl + Alt + J	Surround with Live Template
Ctrl + J	Insert Live Template

General

Alt + #[0-9]	Open corresponding tool window
Ctrl + S	Save all
Ctrl + Alt + Y	Synchronize
Ctrl + Shift + F12	Toggle maximizing editor
Alt + Shift + F	Add to Favorites
Alt + Shift + I	Inspect current file with current profile
Ctrl + BackQuote (`)	Quick switch current scheme
Ctrl + Alt + S	Open Settings dialog
Ctrl + Shift + A	Find Action
Ctrl + Tab	Switch between tabs and tool window

To find any action inside the IDE use Find Action (Ctrl+Shift+A)



jetbrains.com/pycharm



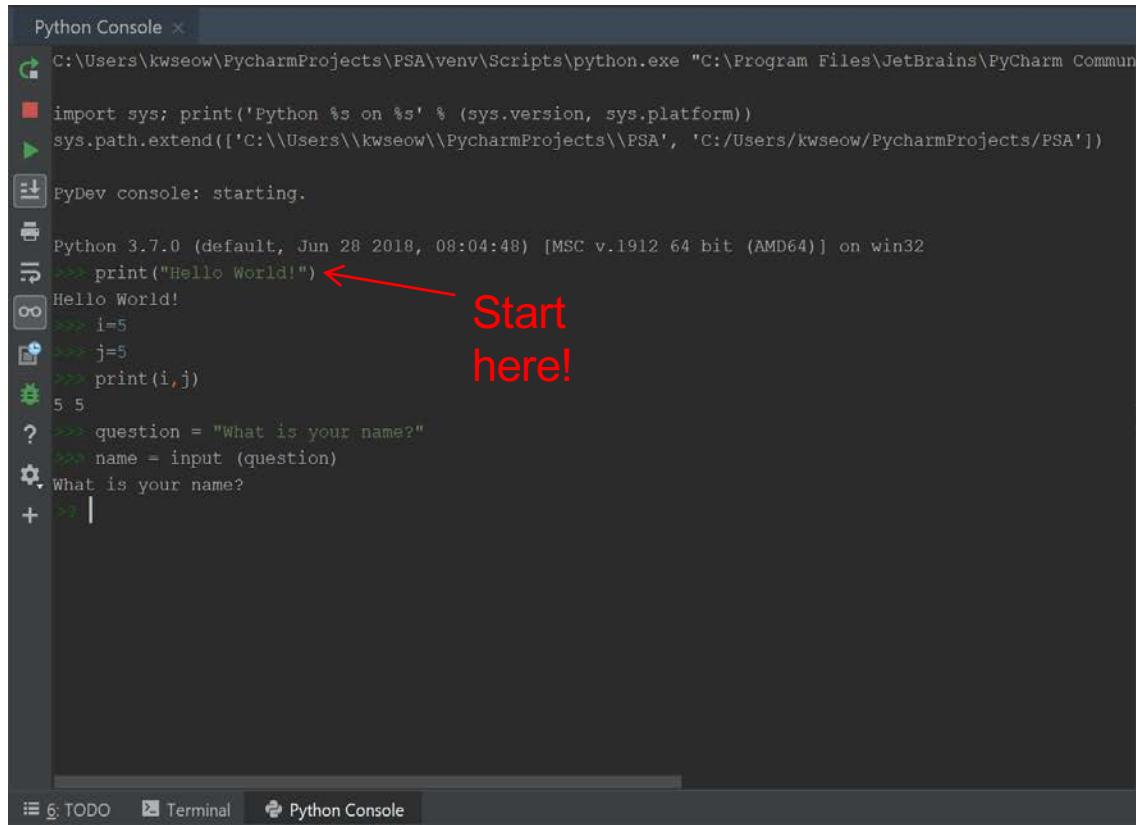
blog.jetbrains.com/pycharm



@pycharm

Using the console

- Also known as the interpreter
- See the output straightaway
- Usually used to test very small chunks of code
- Type code after >>>
- Let's try!



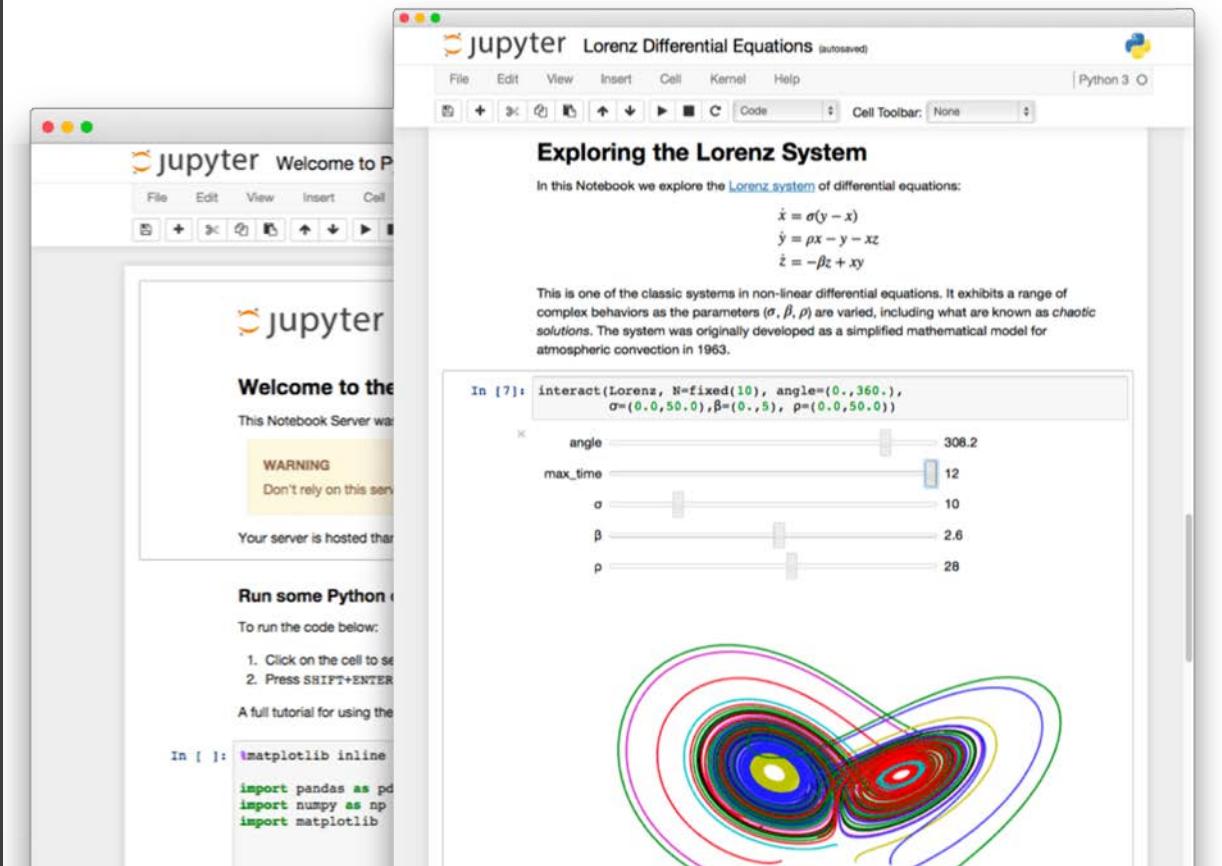
The screenshot shows the PyCharm Python Console window. It displays the following session:

```
C:\Users\kwseow\PycharmProjects\PSA\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2018.1.3\helpers\pydev\pydevconsole.py" -c "import sys; print('Python %s on %s' % (sys.version, sys.platform))\nsys.path.extend(['C:\\\\Users\\\\kwseow\\\\PycharmProjects\\\\PSA', 'C:/Users/kwseow/PycharmProjects/PSA'])"\n\nPyDev console: starting.\n\nPython 3.7.0 (default, Jun 28 2018, 08:04:48) [MSC v.1912 64 bit (AMD64)] on win32\n>>> print("Hello World!")\nHello World!\n>>> i=5\n>>> j=5\n>>> print(i,j)\n5 5\n>>> question = "What is your name?"\n>>> name = input (question)\n>>> What is your name?\n+ >>> |
```

A red arrow points to the line `>>> print("Hello World!")`, and the text "Start here!" is overlaid in red on the right side of the arrow.

Using Jupyter Notebook

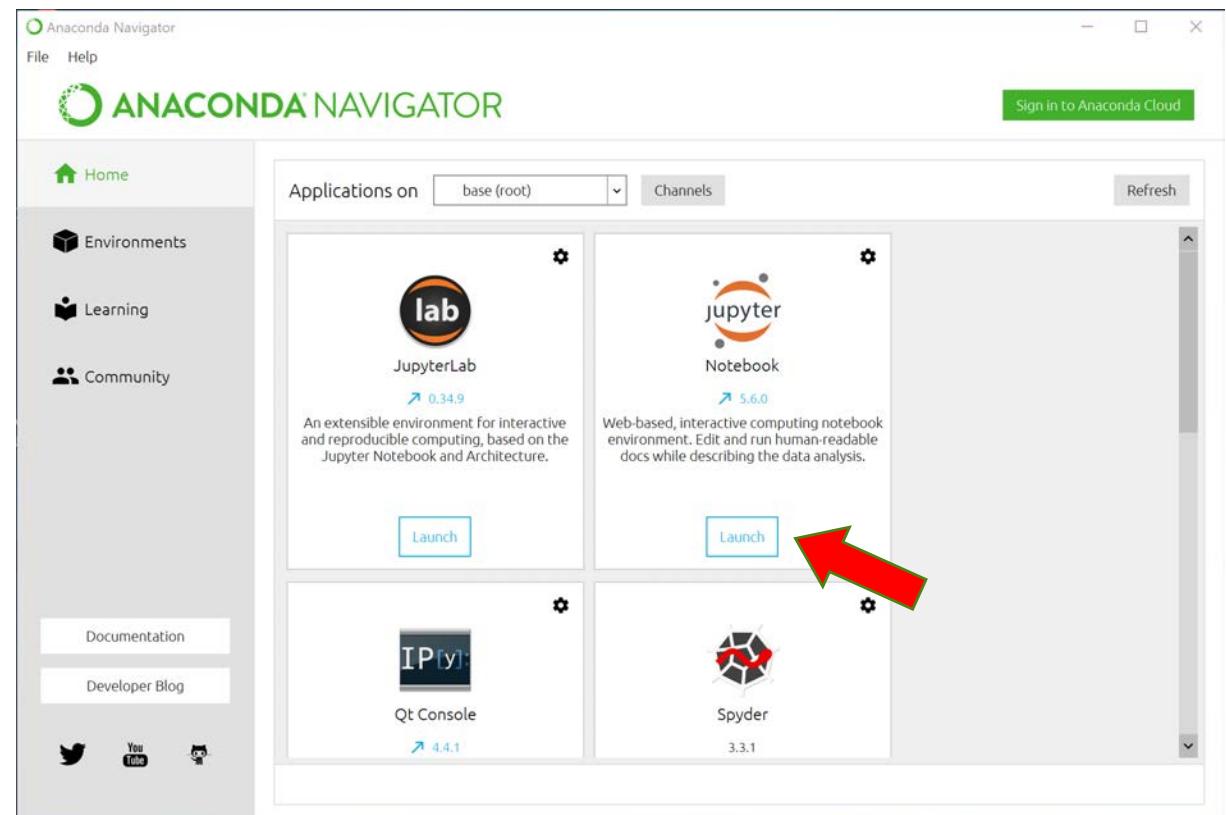
The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



Using Jupyter Notebook

Launch Anaconda Navigator from the start menu

Click on Jupyter Notebook to launch Jupyter

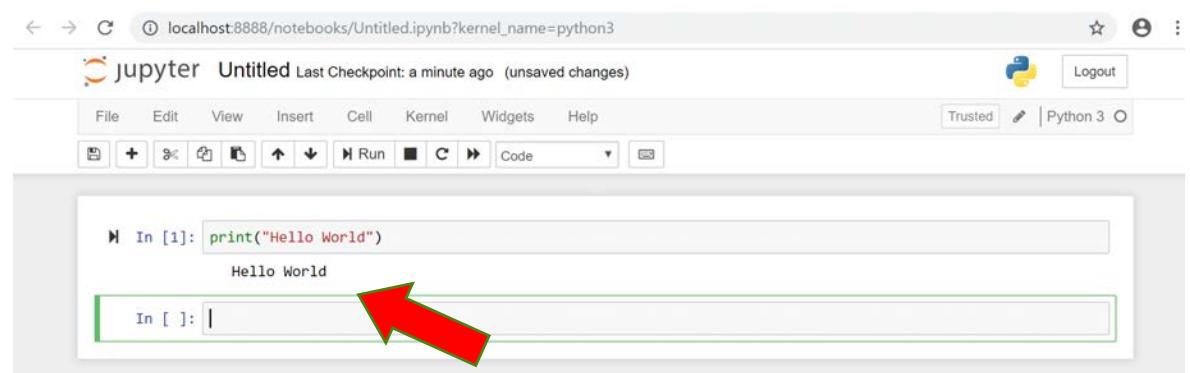
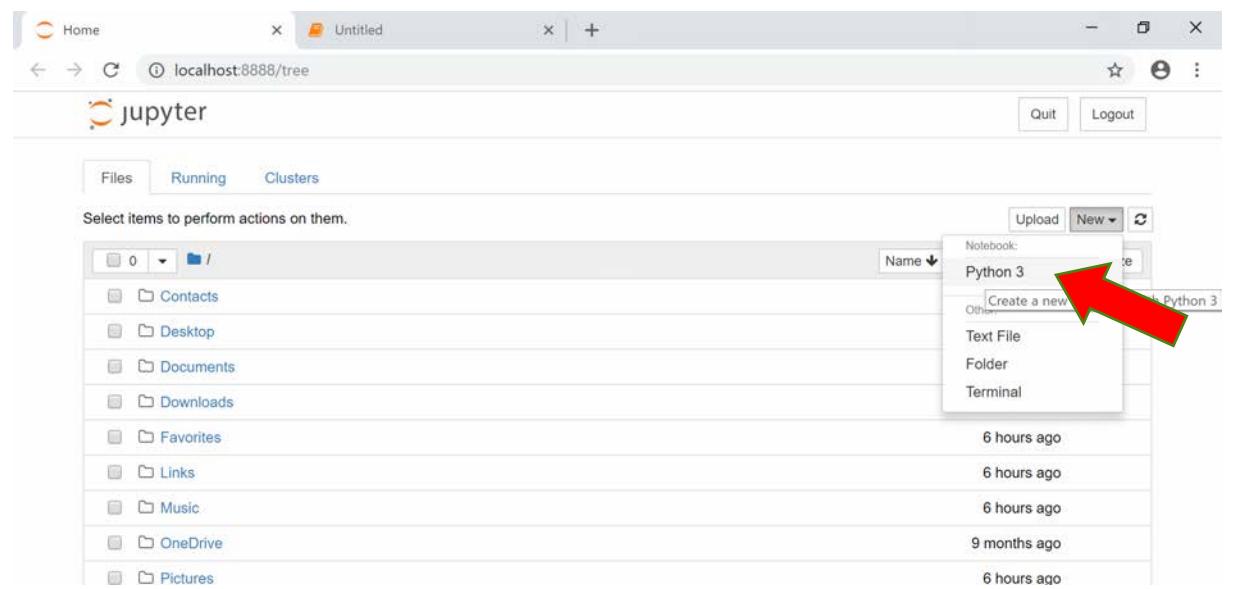


Using Jupyter Notebook

Notebook will launch with your default web browser.

Use New to start a new notebook

To run a “cell”, use shift-enter



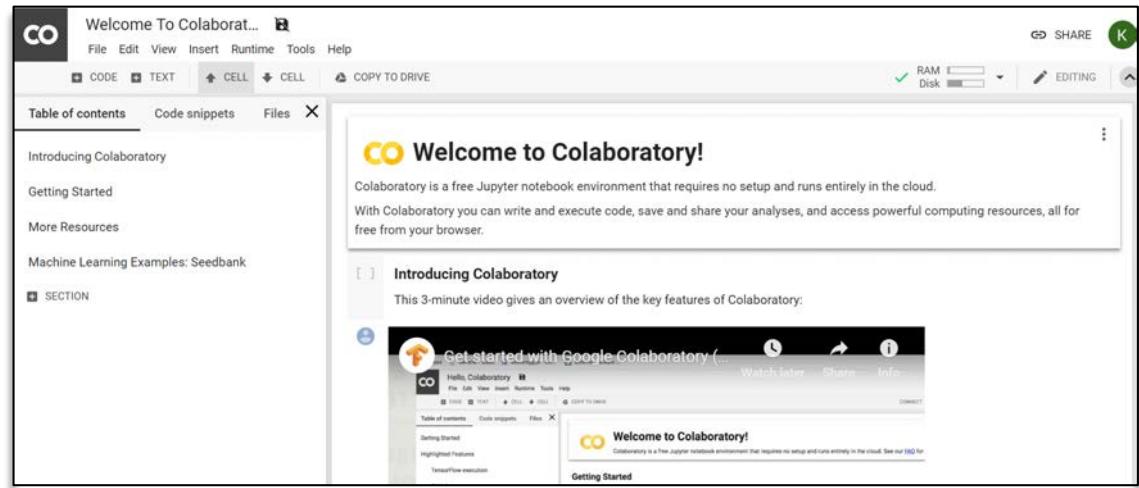
Online Jupyter Notebook

Google:

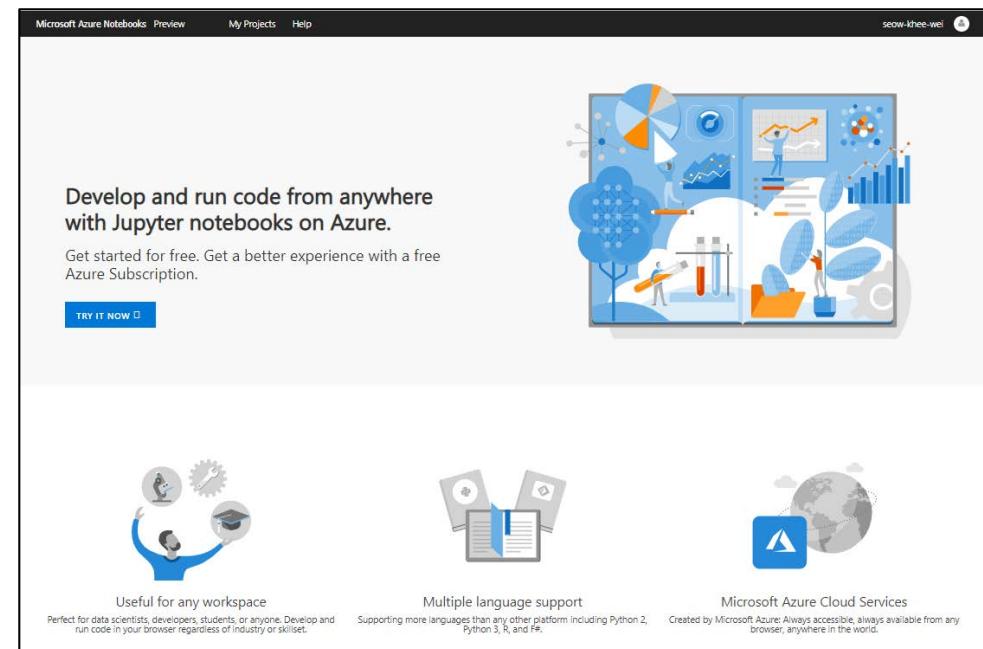
<https://colab.research.google.com>

Microsoft:

<https://notebooks.azure.com/>



The screenshot shows the Google Colaboratory interface. The top navigation bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the bar are buttons for 'CODE', 'TEXT', 'CELL', 'COPY TO DRIVE', and 'RAM Disk'. A sidebar on the left contains links for 'Introducing Colaboratory', 'Getting Started', 'More Resources', and 'Machine Learning Examples: Seedbank'. The main content area displays a video thumbnail titled 'Get started with Google Colaboratory' and a text box with the heading 'Welcome to Colaboratory!'.



The screenshot shows the Microsoft Azure Notebooks landing page. It features a large central image illustrating data analysis and machine learning tasks. Text on the page reads: 'Develop and run code from anywhere with Jupyter notebooks on Azure.' and 'Get started for free. Get a better experience with a free Azure Subscription.' A 'TRY IT NOW' button is visible. Below the main image, there are three sections: 'Useful for any workspace', 'Multiple language support', and 'Microsoft Azure Cloud Services'. Each section includes an icon and a brief description.

Python For Data Science Cheat Sheet

Jupyter Notebook

Learn More Python for Data Science [Interactively](#) at [www.DataCamp.com](#)



Saving/Loading Notebooks

Create new notebook



Make a copy of the current notebook



Save current notebook and record checkpoint



Preview of the printed notebook



Close notebook & stop running any scripts

Open an existing notebook

Rename notebook

Revert notebook to a previous checkpoint

Download notebook as
 - IPython notebook
 - Python
 - HTML
 - Markdown
 - reST
 - LaTeX
 - PDF

Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Edit Cells

Cut currently selected cells to clipboard



Paste cells from clipboard above current cell

Copy cells from clipboard to current cursor position

Paste cells from clipboard on top of current cell

Paste cells from clipboard below current cell

Revert "Delete Cells" invocation

Delete current cells

Merge current cell with the one above

Split up a cell from current cursor position

Move current cell up

Merge current cell with the one below

Adjust metadata underlying the current notebook

Move current cell down

Remove cell attachments

Find and replace in selected cells

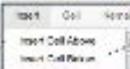
Paste attachments of current cell

Copy attachments of current cell

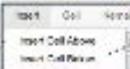
Insert Cells

Insert image in selected cells

Add new cell above the current one



Add new cell below the current one



Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:



IPython



R kernel



Julia

Installing Jupyter Notebook will automatically install the IPython kernel.

Restart kernel

Restart kernel & run all cells

Restart kernel & run all cells



Interrupt kernel

Interrupt kernel & clear all output

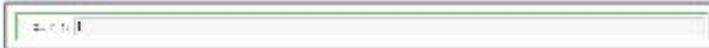
Connect back to a remote notebook

Run other installed kernels

Command Mode:



Edit Mode:



Executing Cells

Run selected cell(s)



Run current cells down and create a new one above

Run current cells down and create a new one below

Run all cells above the current cell

Run all cells

Change the cell type of current cell

Run all cells below the current cell

toggle, toggle scrolling and clear current outputs

toggle, toggle scrolling and clear current outputs

View Cells

Toggle display of Jupyter logo and filename



Toggle line numbers in cells

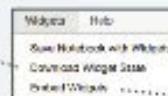
Toggle display of toolbar

Toggle display of cell action icons:
 - None
 - Edit metadata
 - Raw cell format
 - Slideshow
 - Attachments
 - Tags

Widgets

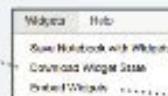
Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.



Save notebook with interactive widgets
 Embed current widgets

Download serialized state of all widget models in use



1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cells
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

Asking For Help

Walk through a UI tour

Edit the built-in keyboard shortcuts



List of built-in keyboard shortcuts

Description of markdown available in notebook



Notebook help topics

Python help topics



Information on unofficial Jupyter Notebook extensions

NumPy help topics



IPython help topics

Matplotlib help topics



SciPy help topics

Pandas help topics



Sympy help topics

About Jupyter Notebook



About Jupyter Notebook



Others

IOS:

<http://omz-software.com/pythonista/>

BeeWare:

<https://pybee.org/>

Pythonista 3

A Full Python IDE for iOS

Pythonista is a complete development environment for writing Python™ scripts on your iPad or iPhone. Lots of examples are included – from games and animations to plotting, image manipulation, custom user interfaces, and automation scripts.

In addition to the powerful standard library, Pythonista provides extensive support for interacting with native iOS features, like contacts, reminders, photos, location data, and more.



BeeWare Project News Community Contributing Contact Donate



BeeWare

Build native apps with Python.

Write your apps in Python and release them on iOS, Android, Windows, MacOS, Linux, Web, and tvOS using rich, native user interfaces. One codebase. Multiple apps.

[Take the Tutorial](#) [I Want To Contribute](#)

[Donate and Support Us!](#)

Data Types

We shall focus on these basic data types in our workshop:

Numbers

int for whole numbers

float for numbers with decimal point, e.g. 5.2, 2.0

Text

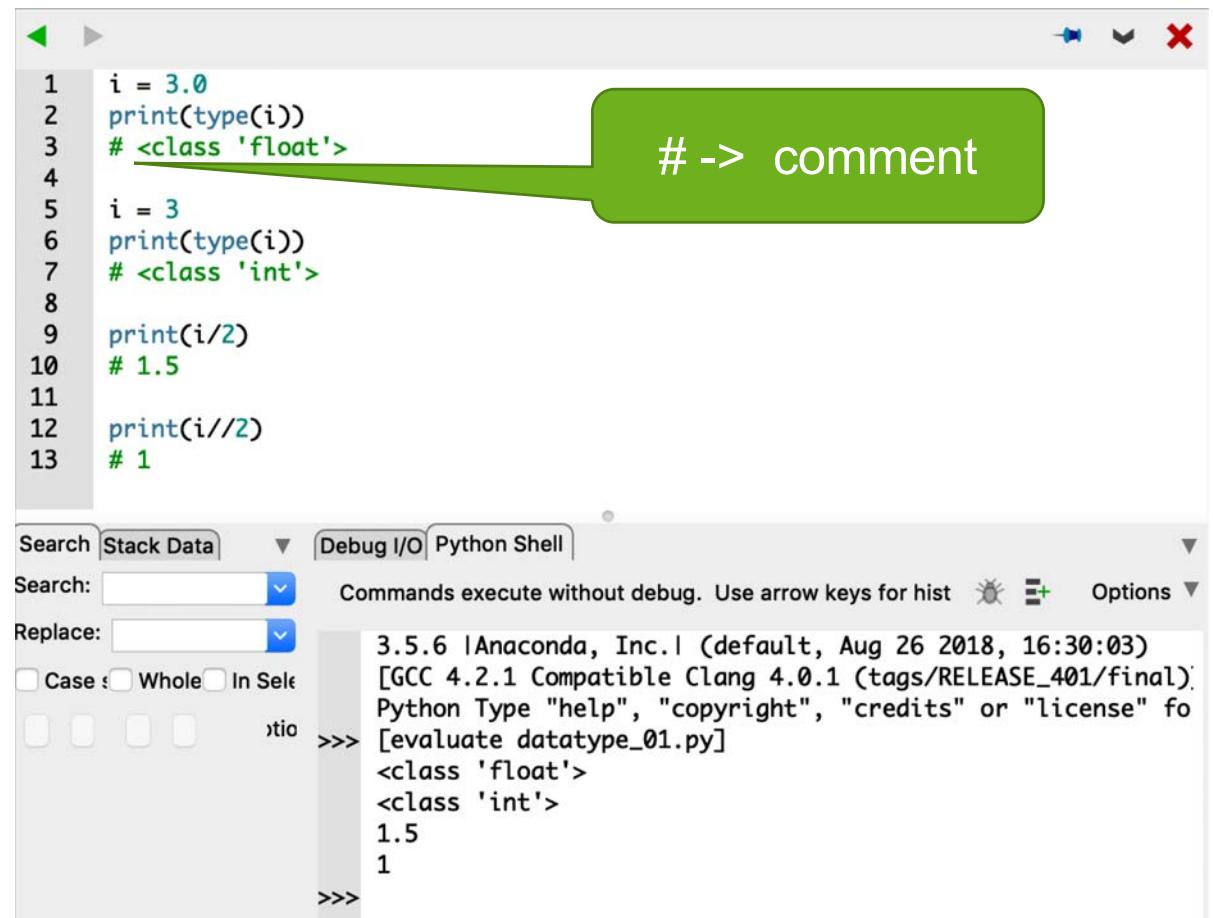
str for a sequence of characters

Containers

list a sequence of objects, use an index to access each object

Data Types - Numbers

- Python is ‘friendly’ language
- The data type of a variable will change automatically depending on the values assigned to it.
- i is a variable
- In the 1st case, a float number is assigned to a variable i, so its data type is float.
- In the 2nd case, if an integer value is assigned to the same variable, its data type will change to integer.
- Use # to add comments to your code



```

1 i = 3.0
2 print(type(i))
3 # <class 'float'>
4
5 i = 3
6 print(type(i))
7 # <class 'int'>
8
9 print(i/2)
10 # 1.5
11
12 print(i//2)
13 # 1

```

The screenshot shows a Python development environment with the following details:

- Code Editor:** Displays the provided Python script. The first two lines define a variable `i` with values `3.0` and `3` respectively, and then print their types. The third line contains a comment `# <class 'float'>`. The last three lines perform division and floor division operations on `i`, resulting in outputs `1.5` and `1` respectively.
- Python Shell:** Located at the bottom right, shows the Python interpreter session. It includes the Python version (`3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)`), compiler information (`[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)`), and the results of the script execution.
- Annotations:** A green callout bubble with the text `# -> comment` points to the `#` character in the third line of the code editor.

Data Types - Strings

- Strings contain characters
- Strings can be added together with the + operator
- Strings can contain number characters, but they are not numbers (int or float)
- You can not add a number to a string
- The * operator will replicate the string with an integer value

```

1 s = "hello"
2 t = "world"
3 print(s + " " + t)
4 # hello world
5
6 s = "123"
7 print(s * 4)
8 # 123123123123
9
10 print(s + 4)
11 # error!

```

Search Stack Data Debug I/O Python Shell

Search: Replace: Case Whole In Selection

Commands execute without debug. Use arrow keys for hist Options

3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for more information
>>> [evaluate datatype_01.py]
hello world
123123123123
Traceback (most recent call last):
 File "/Users/kwseow/Dropbox/Projects/V7.PSA/Day1Resource.py", line 10, in <module>
 print(s + 4)
builtins.TypeError: Can't convert 'int' object to str implicitly

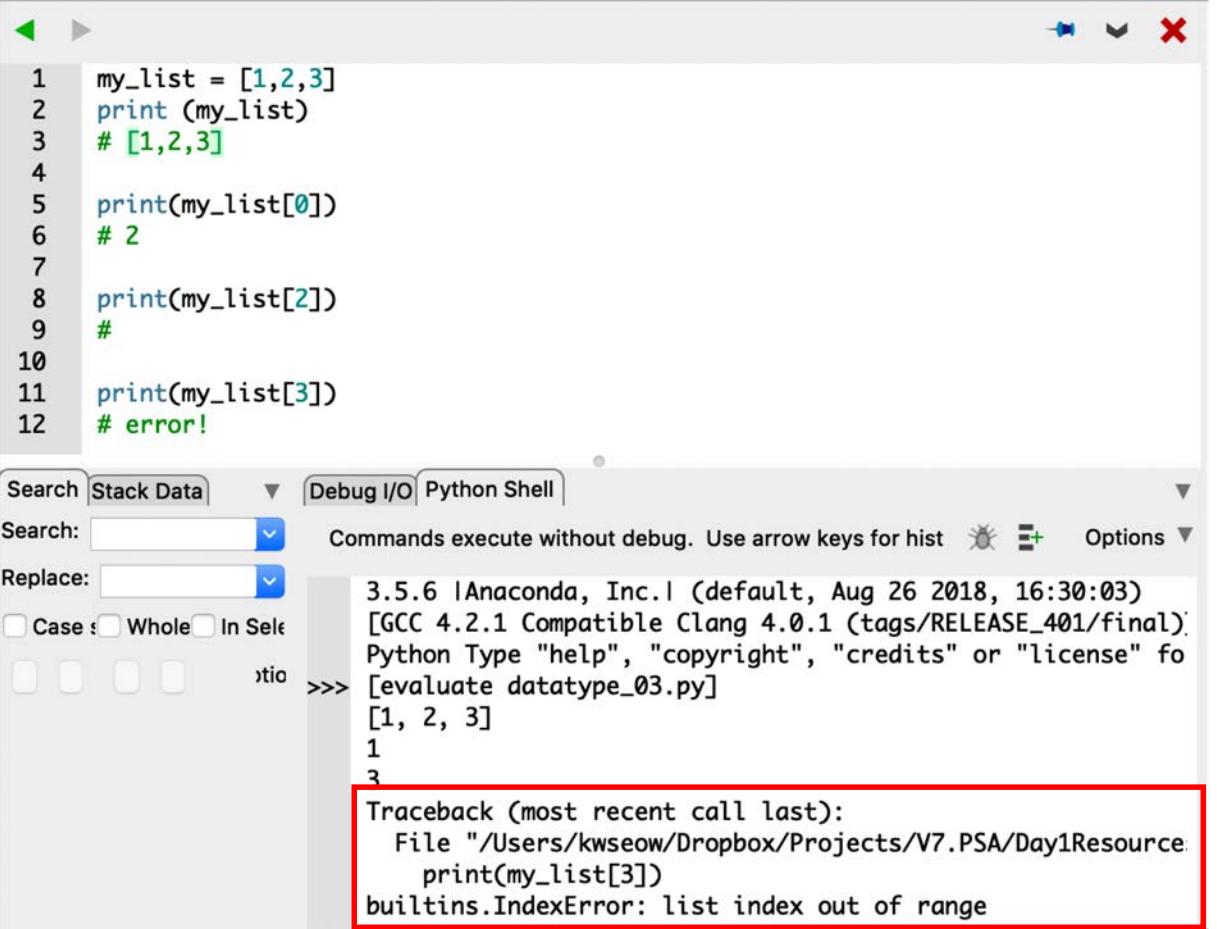
Notes:

Strings are Unicode by default for Python 3

For string assignment, you can use “hello” or ‘hello’ but not “hello”

Data Types - List

- List is a collection of objects
- Lists can contain any type of objects
- Items in a list must be accessed by an index
- First index position starts from 0
- Python doesn't like it if you ask for something that is not in the list
- Try using a negative index, e.g. -1: What happens?



The screenshot shows a Python code editor with the following code:

```

1 my_list = [1,2,3]
2 print (my_list)
3 # [1,2,3]
4
5 print(my_list[0])
6 # 2
7
8 print(my_list[2])
9 #
10 print(my_list[3])
11 # error!

```

Below the code, the Python Shell tab is active, displaying the following output:

```

3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" fo
>>> [evaluate datatype_03.py]
[1, 2, 3]
1
3

```

A red box highlights the final line of the traceback:

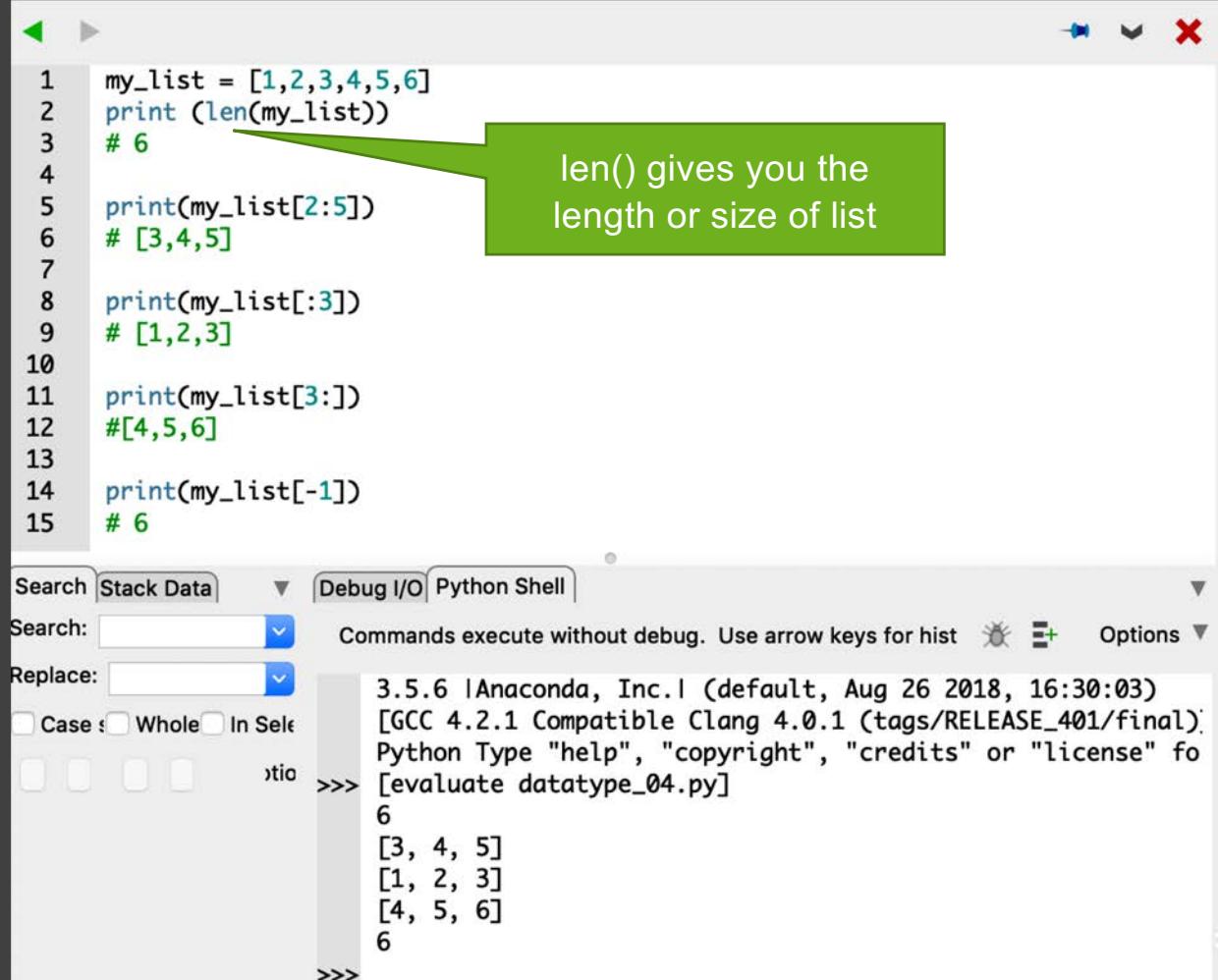
```

Traceback (most recent call last):
  File "/Users/kwseow/Dropbox/Projects/V7.PSA/Day1Resource
    print(my_list[3])
builtins.IndexError: list index out of range

```

Data Types - List

- len() gives you the length or size of list
- Get a range of items using : colon
(This is called slicing)
 - [$:3$] first 3 items
 - [$3:$] last 3 items
- Negative index gives you items from the back
 - [$-x$] x^{th} last item



The screenshot shows a Python code editor with the following code:

```

1 my_list = [1,2,3,4,5,6]
2 print(len(my_list))
3 # 6
4
5 print(my_list[2:5])
6 # [3,4,5]
7
8 print(my_list[:3])
9 # [1,2,3]
10
11 print(my_list[3:])
12 # [4,5,6]
13
14 print(my_list[-1])
15 # 6

```

A green callout box points to the first print statement with the text: "len() gives you the length or size of list".

Below the code editor is a Python shell window showing the execution results:

```

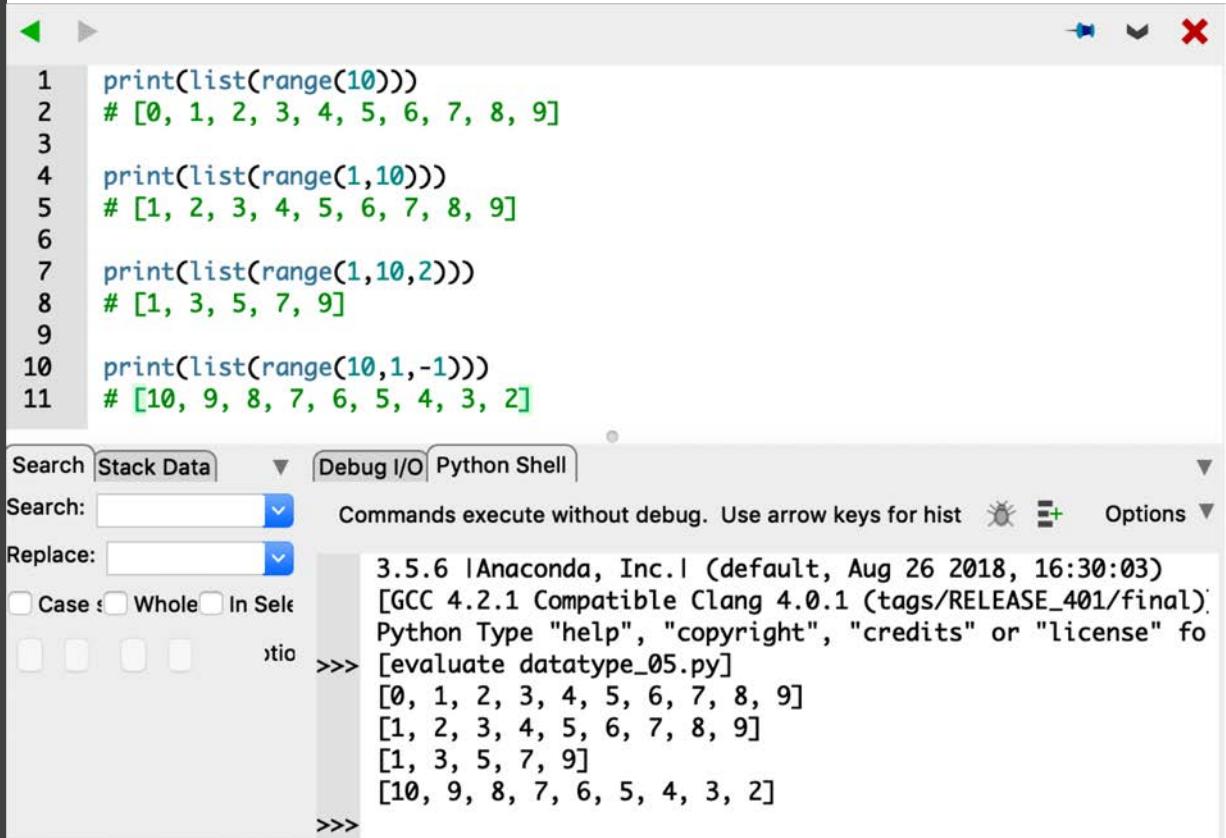
Search Stack Data Debug I/O Python Shell
Search: Replace: Commands execute without debug. Use arrow keys for hist Options
Case Whole In Selection
>>> 3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for
[evaluate datatype_04.py]
6
[3, 4, 5]
[1, 2, 3]
[4, 5, 6]
6
>>>

```

Range

Three versions:

- `range(y)`
 - starts at 0
 - ends before y
 - step up by 1
- `range(x, y)`
 - starts at x
 - ends before y
 - step up by 1
- `range(x, y, s)`
 - starts at x
 - ends before y
 - step up by s



```

1 print(list(range(10)))
2 # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3
4 print(list(range(1,10)))
5 # [1, 2, 3, 4, 5, 6, 7, 8, 9]
6
7 print(list(range(1,10,2)))
8 # [1, 3, 5, 7, 9]
9
10 print(list(range(10,1,-1)))
11 # [10, 9, 8, 7, 6, 5, 4, 3, 2]

```

The screenshot shows a Python terminal window with the following content:

Search Stack Data Debug I/O Python Shell

Search: Replace: Commands execute without debug. Use arrow keys for hist Options

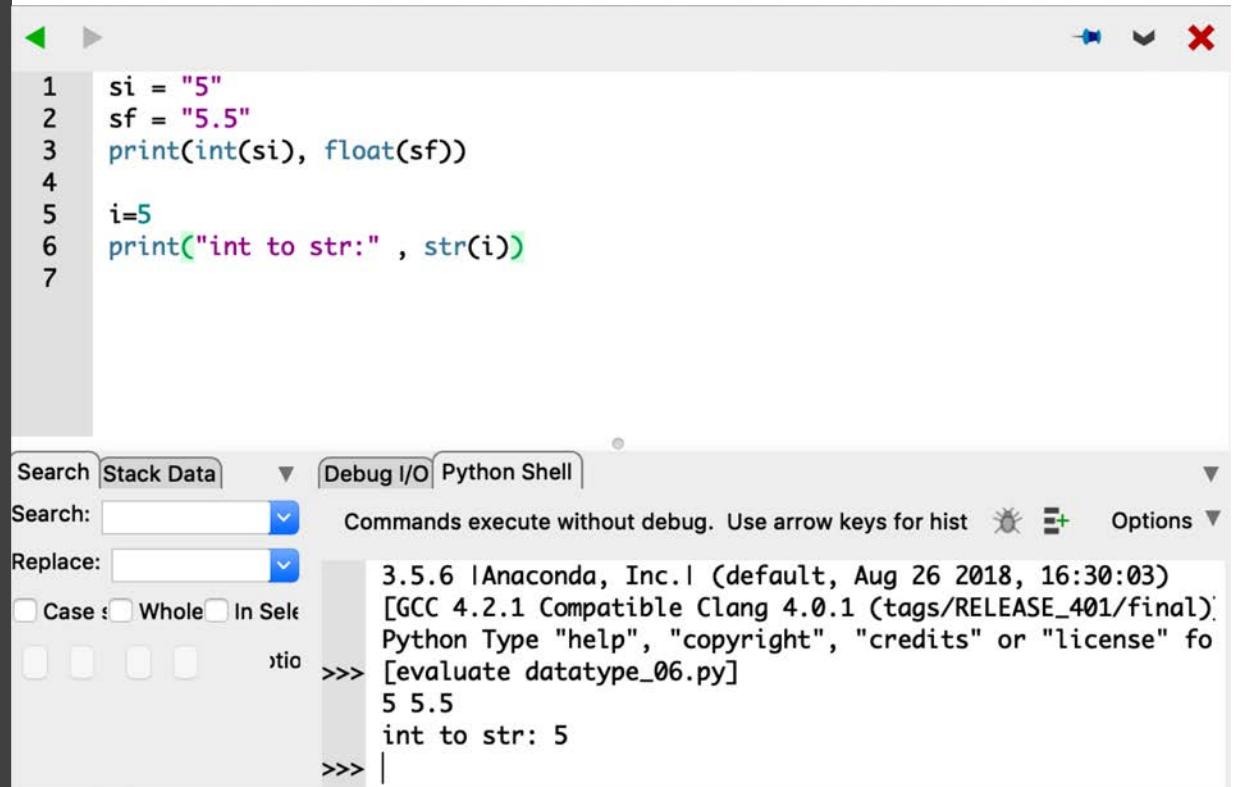
Case Whole In Selection

>>> [evaluate datatype_05.py]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2]

Note: if s is negative, then step down by its absolute value

Data Types - Conversion

- You can convert anything into its string version:
`str(...)`
- A string containing all digit characters a decimal point can be converted into a number type:
`int(...)` or `float(...)`



The screenshot shows a Python IDE interface with a code editor and a Python Shell window.

Code Editor:

```
1 si = "5"
2 sf = "5.5"
3 print(int(si), float(sf))
4
5 i=5
6 print("int to str: " , str(i))
```

Python Shell:

Search Stack Data Debug I/O Python Shell

Search: Replace: Commands execute without debug. Use arrow keys for hist Options

< Case < Whole < In Selection

```
>>> [evaluate datatype_06.py]
5 5.5
int to str: 5
>>> |
```

Print Formatted Numbers

Formatting numbers

%d	int
%f	float
%s	string

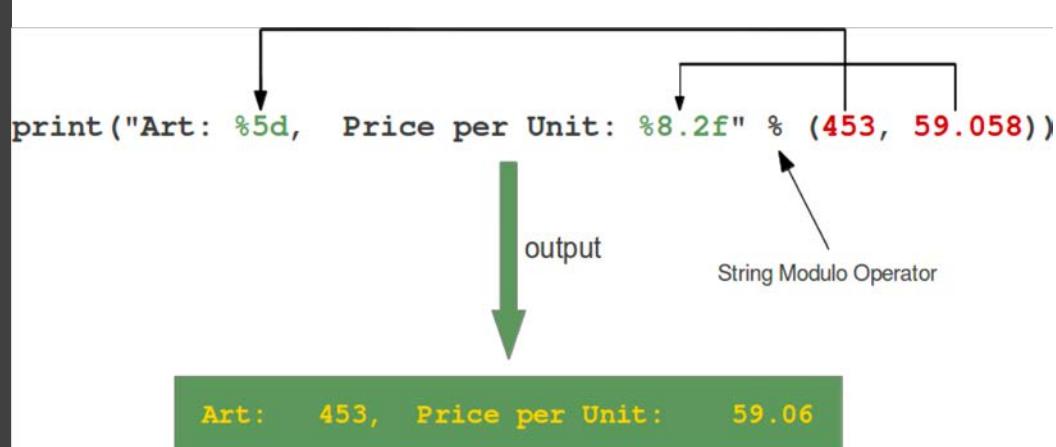
Special formatting

%.2f

- float
- two digits behind the point

%+d (or f)

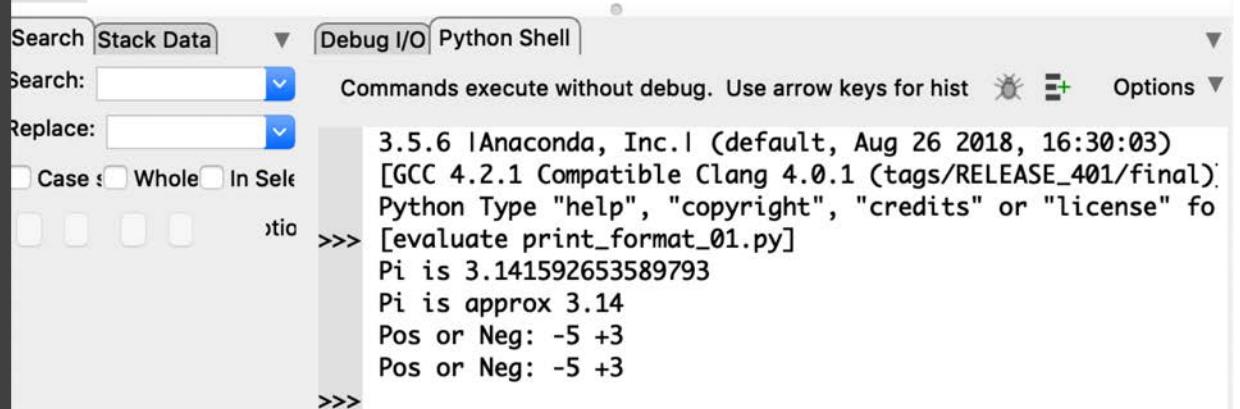
- force print the sign



```

1 import math
2 print("Pi is " + str (math.pi))
3 # Pi is 3.141592653589793
4
5 print("Pi is approx %0.2f" % (math.pi))
6 # Pi is approx 3.14
7
8 print("Pos or Neg: %+d %+d" % (-5,3))
9 # Pos or Neg: -5 +3
10
11 tmp_str = "Pos or Neg"
12 print("%s: %+d %+d" % (tmp_str,-5,3))
13 # Pos or Neg: -5 +3

```



Search Stack Data Debug I/O Python Shell

Search: Replace: Case Whole In Selection

Commands execute without debug. Use arrow keys for hist Options

3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for
>>> [evaluate print_format_01.py]
Pi is 3.141592653589793
Pi is approx 3.14
Pos or Neg: -5 +3
Pos or Neg: -5 +3

Functions

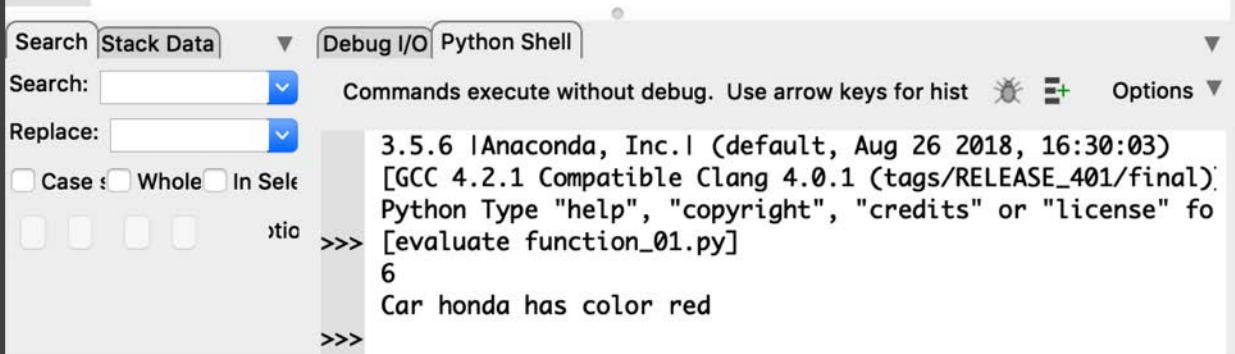
Passing parameters

- Arguments must be passed in order

Alternatively, use parameter names to identify the arguments

Be sure to have
indentation

```
identCar
1 def multiply(a,b):
2     return a*b
3
4 print(multiply(2,3))
5
6 def identCar(car,colour):
7     return "Car %s has color %s" % (car, colour)
8
9 print(identCar(colour="red", car="honda"))
10
```

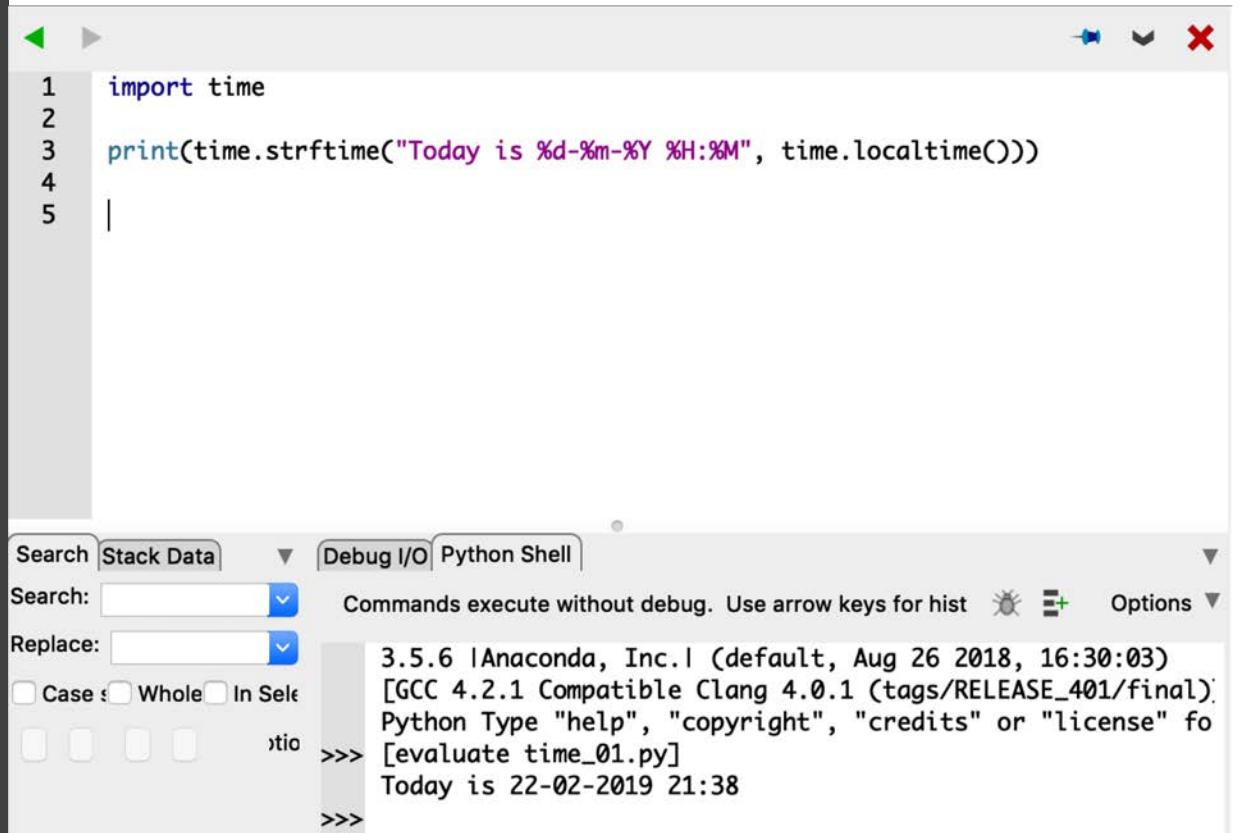


The screenshot shows a Python IDE interface with a code editor and a Python Shell tab. The code editor contains the same Python script as above. The Python Shell tab shows the following output:

```
Search Stack Data Debug I/O Python Shell
Search: Replace: Commands execute without debug. Use arrow keys for hist Options
Case Whole In Selection
3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for
[evaluate function_01.py]
6
Car honda has color red
```

The time library

One of the functions in the time library is **strftime**, a flexible function to display the time based on certain format.



```
import time
print(time.strftime("Today is %d-%m-%Y %H:%M", time.localtime()))
|
```

Search Stack Data Debug I/O Python Shell

Search: Replace: Commands execute without debug. Use arrow keys for hist Options

< Case < Whole < In Selection

>>> 3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for
>>> [evaluate time_01.py]
Today is 22-02-2019 21:38

[https://docs.python.org/3/library/time.html?
highlight=strftime#time.strftime](https://docs.python.org/3/library/time.html?highlight=strftime#time.strftime)

The datetime library

The datetime module allows manipulating dates and times in both simple and complex ways. date and time arithmetic is supported

```
1  from datetime import datetime, timedelta
2  d1 = datetime(1991, 4, 30)
3  print(d1)
4  # -> 1991-04-30 00:00:00
5
6  d2 = d1 + timedelta(10)
7  print(d2)
8  # -> 1991-05-10 00:00:00
9
10 print(d2 - d1)
11 # -> 10 days, 0:00:00
12
13 d3 = d1 - timedelta(100)
14 print(d3)
15 # -> 1991-01-20 00:00:00
16
17 d4 = d1 - 2 * timedelta(50)
18 print(d4)
19
20 #adding seconds
21 d1 = datetime(1991, 4, 30)
22 print(d1)
23 # -> 1991-04-30 00:00:00
24
25 d2 = d1 + timedelta(10,100)
26 print(d2)
27 # -> 1991-05-10 00:01:40
28
29 print(d2 - d1)
30 # -> 10 days, 0:01:40
31
32 #get current Date
33 d1 = datetime.now()
34 print(d1)
35 # -> 2019-03-07 12:51:51.196327
36 print(d1.day, d1.hour, d1.second)
37 # -> 2019-03-07 12:51:51.196327
```

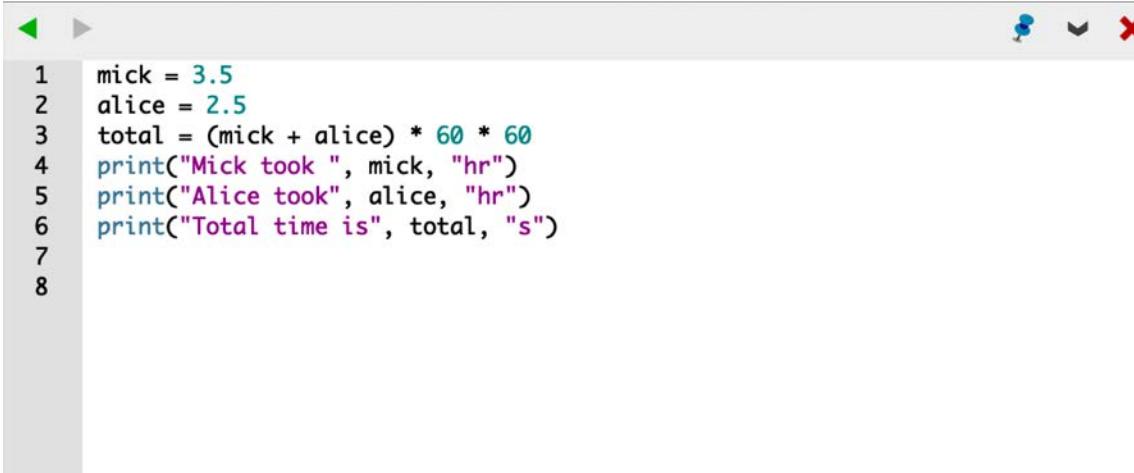
Basic Arithmetic

Operator Name	Code	Example When x = 2 and y = 1
Plus	$x + y$	$x + y$ will give 3.
Minus	$x - y$	$x - y$ will give 1.
Divide	x / y	x / y will give 2.
Multiply	$x * y$	$x * y$ will give 2. You must use * instead of x for multiplication.
x to the power of y	$x ** y$	$x ** y$ means 2 to power of 1 and will give 2.
Modulus	$x \% y$	$x \% y$ will give 0. 0 is the remainder from 2 divides by 1.

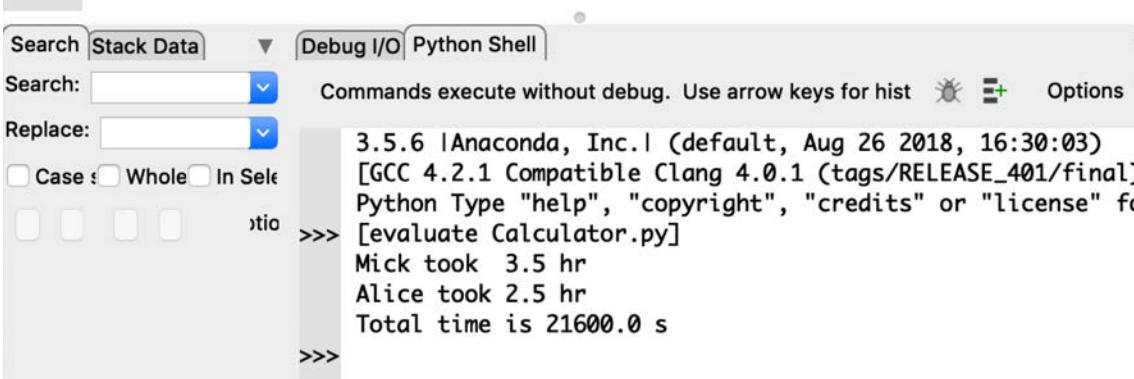
Exercise

Homework Calculator

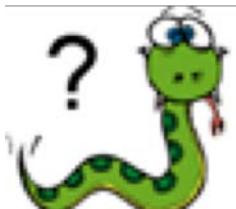
- Mick took 3.5 hours to finish his homework. Alice took 2.5 hours to finish her homework. Write a program to calculate the total amount of time in seconds that they took to finish their homework



```
mick = 3.5
alice = 2.5
total = (mick + alice) * 60 * 60
print("Mick took ", mick, "hr")
print("Alice took", alice, "hr")
print("Total time is", total, "s")
```



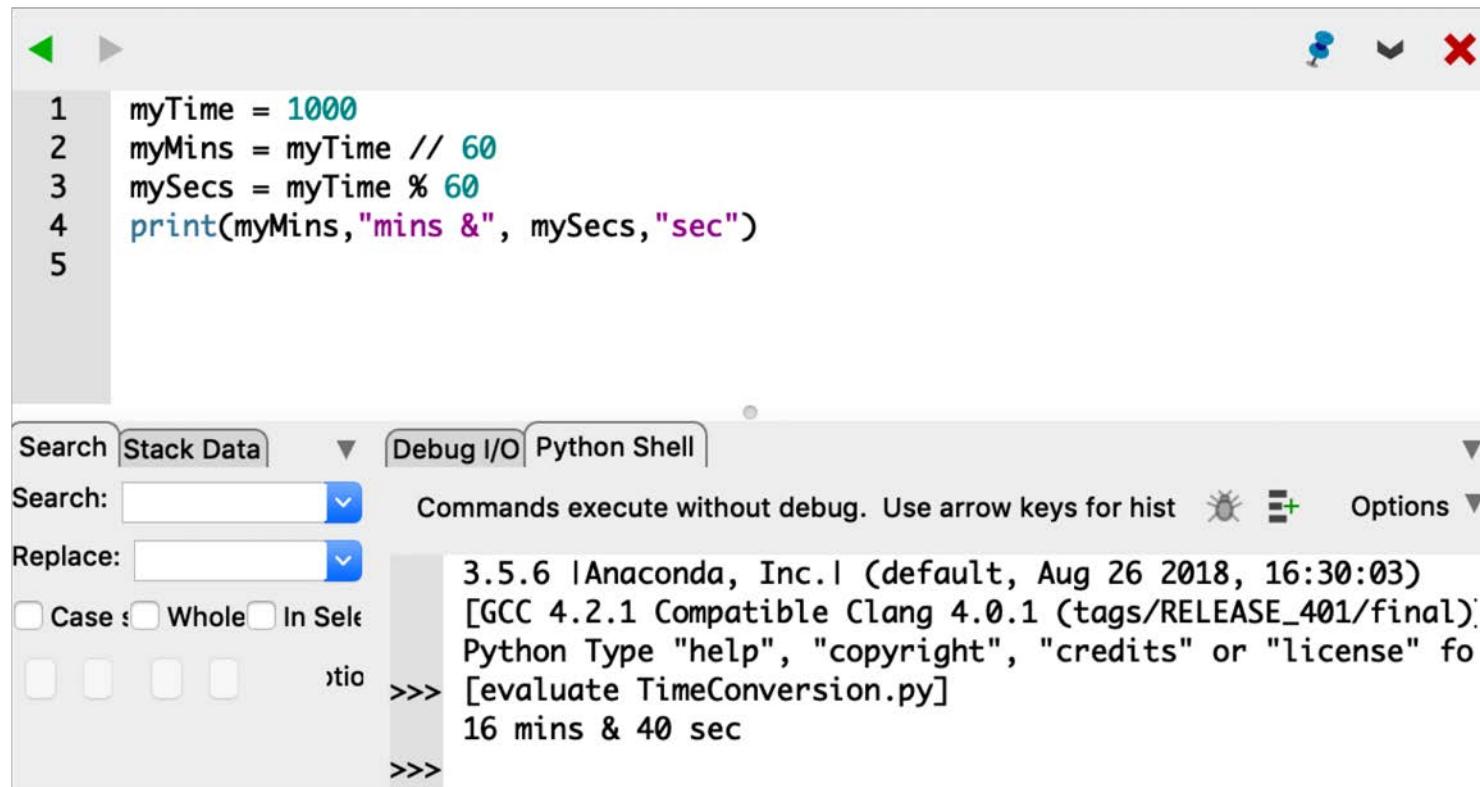
```
3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for
[evaluate Calculator.py]
Mick took 3.5 hr
Alice took 2.5 hr
Total time is 21600.0 s
```



Exercise

Time Conversion

- Write a program (in 1 script file) to convert 1000 seconds to minutes and seconds.

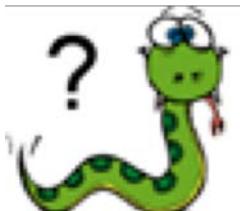


A screenshot of a Python code editor and terminal window. The code editor shows a script named TimeConversion.py with the following content:

```
1 myTime = 1000
2 myMins = myTime // 60
3 mySecs = myTime % 60
4 print(myMins,"mins &", mySecs,"sec")
5
```

The terminal window below shows the output of running the script:

```
3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" fo
[evaluate TimeConversion.py]
16 mins & 40 sec
```

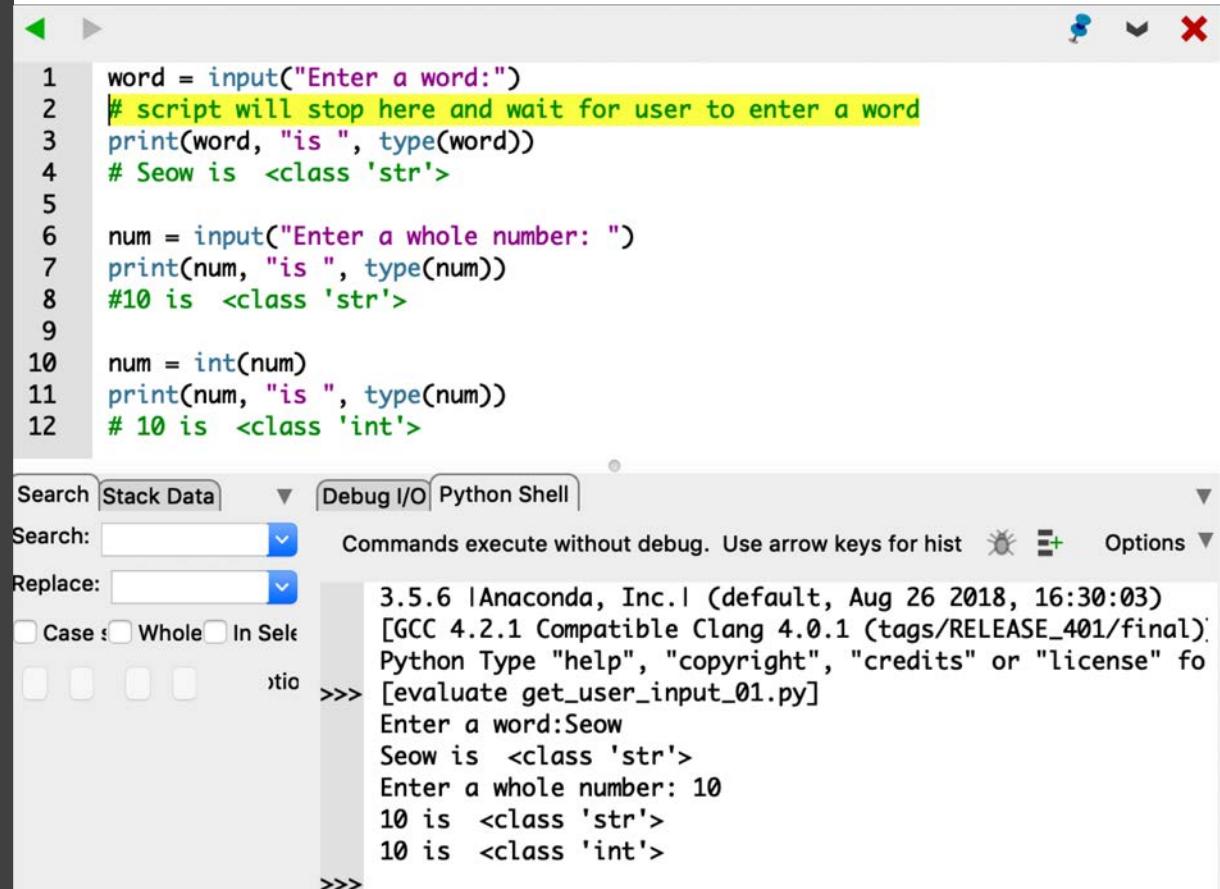


Getting User Input

You can use `input()` function to ask for user input.

The value entered by the user is stored into a variable as a string.

If the value is to be used as a number, you can use the `int()` or `float()` function to convert the value to the appropriate number data type.



```
1 word = input("Enter a word:")
2 # script will stop here and wait for user to enter a word
3 print(word, "is ", type(word))
4 # Seow is <class 'str'>
5
6 num = input("Enter a whole number: ")
7 print(num, "is ", type(num))
8 #10 is <class 'str'>
9
10 num = int(num)
11 print(num, "is ", type(num))
12 # 10 is <class 'int'>
```

Search Stack Data Debug I/O Python Shell

Search: Replace: Commands execute without debug. Use arrow keys for hist Options

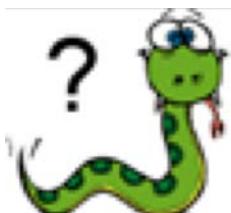
3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" fo
[evaluate get_user_input_01.py]
Enter a word:Seow
Seow is <class 'str'>
Enter a whole number: 10
10 is <class 'str'>
10 is <class 'int'>

Exercise

Temperature Calculator

- The normal human body temperature is 36.9 Degree Celsius. Write a program to ask the user for name and temperature and print a message on the screen that indicate the temperature difference from the normal body temperature.

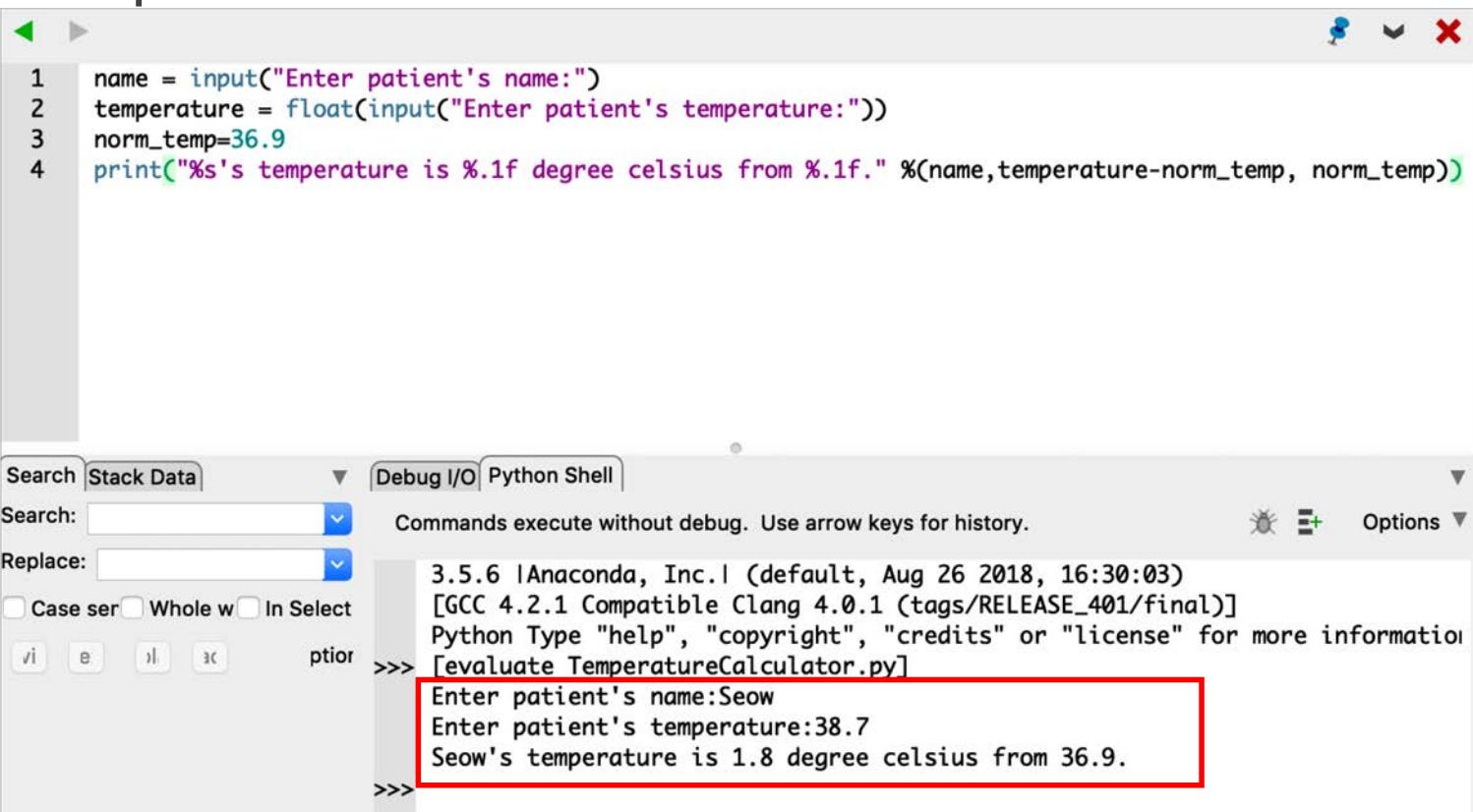
```
Enter patient's name:Seow
Enter patient's temperature:38.7
Seow's temperature is 1.8 degree celsius from 36.9.
```



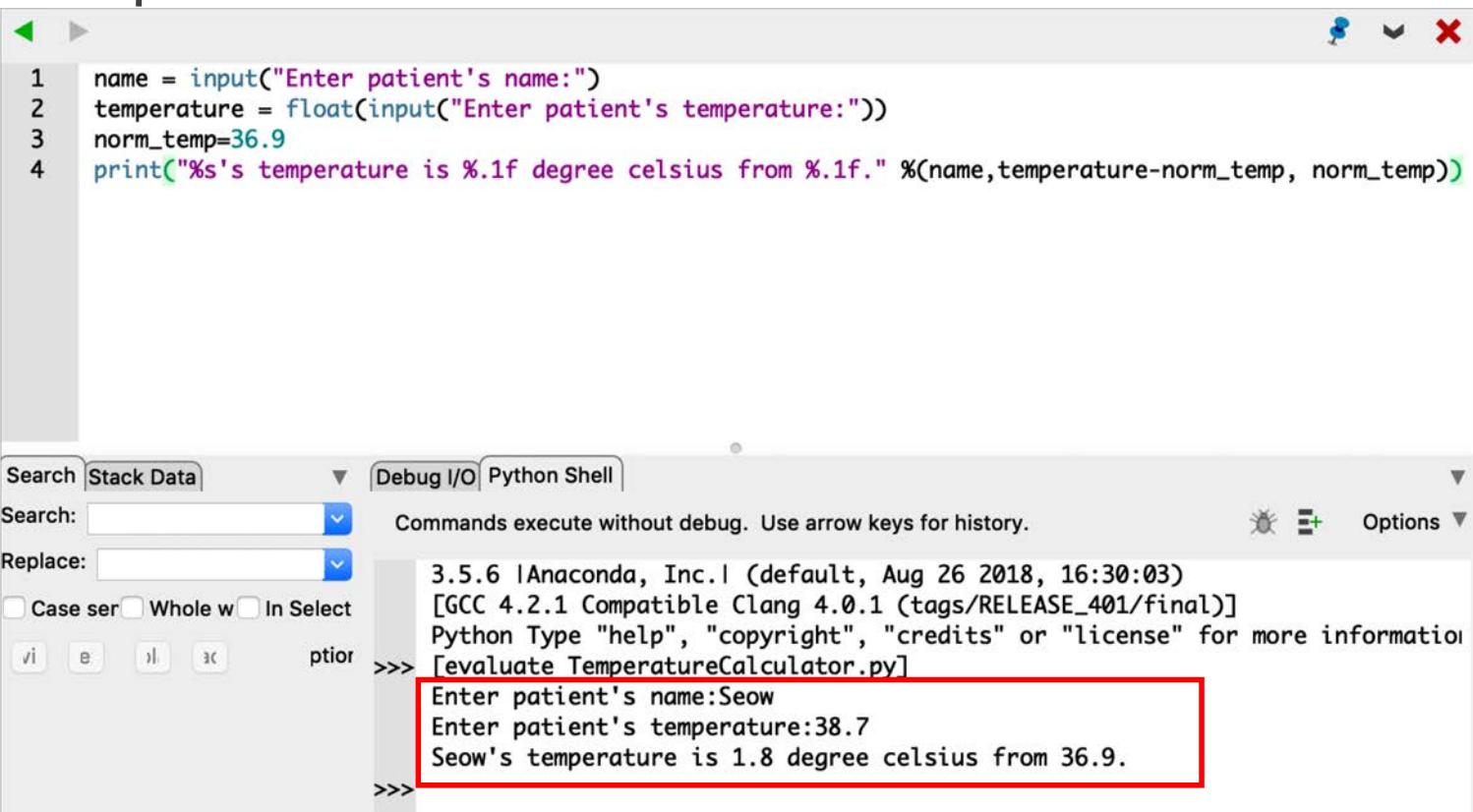
Exercise

Temperature Calculator

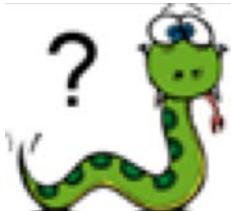
- The normal human body temperature is 36.9 Degree Celsius. Write a program to ask the user for name and temperature and print a message on the screen that indicate the temperature difference from the normal body temperature.



```
name = input("Enter patient's name:")
temperature = float(input("Enter patient's temperature:"))
norm_temp=36.9
print("%s's temperature is %.1f degree celsius from %.1f." %(name,temperature-norm_temp, norm_temp))
```



3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for more information
[evaluate TemperatureCalculator.py]
Enter patient's name:Seow
Enter patient's temperature:38.7
Seow's temperature is 1.8 degree celsius from 36.9.



If-Else Statement

IF STATEMENTS

<https://www.youtube.com/watch?v=fVUL-vzrlcM>

If-Else Statement

```
Enter password:- secret
You have entered correct password
```

```
Enter password:- letsguess
You have entered wrong password
```

```

1 correct_password = "secret"
2 password = input("Enter password:-")
3
4 if password == correct_password:
5     print("You have entered correct password")
6 else:
7     print("You have entered wrong password")
```

All lines, except line 5 are executed as
if password == correct_password
 returns **True**.

All lines, except lines 6-7 are executed as
if password == correct_password
 returns **False**.

Expression	What it does
a == b	Evaluates to True when a is equal to b
a != b	Evaluates to True when a is not equal to b
a < b	Evaluates to True when a is lesser than b
a > b	Evaluates to True when a is bigger than b
a <= b	Evaluates to True when a is lesser than or equal to b
a >= b	Evaluates to True when a is greater than or equal to b

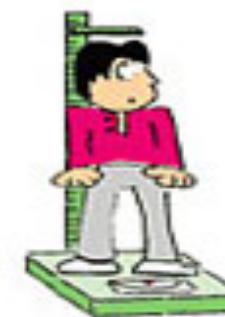
If...elif...else

Type out the code below:

```
1  number1 = input("Enter first number:- ")
2  number2 = input("Enter second number:- ")
3  number1 = float(number1)
4  number2 = float(number2)
5
6  if number1 < number2:
7      print("First number is smaller than the second number.")
8  elif number1 > number2:
9      print("First number is greater than the second number.")
10 else:
11     print("Two numbers are equal.")
```

Exercise BMI Calculator

Develop a BMI Calculator to calculate the BMI of a patient given the weight and height.

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height (m)} \times \text{Height (m)}}$$
A cartoon illustration of a person wearing a pink shirt and grey pants standing on a green scale. To the left of the scale is a vertical height measurement chart with markings from 1 to 2 meters.

Category	Underweight	Ideal	Overweight	Obese
$\text{BMI} = \frac{\text{weight(kg)}}{\text{height(m)}^2}$	< 18	≥ 18 , but < 25	≥ 25 , but < 30	≥ 30



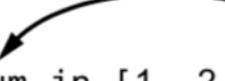
For Loops

- For loops often go hand-in-hand with lists
- Every object in the list will be processed by what is inside the for loop
- What is the data type of *i*?

1st iteration: 
`for num in [1, 2, 3, 4, 5]:
 print(num)`

2nd iteration: 
`for num in [1, 2, 3, 4, 5]:
 print(num)`

3rd iteration: 
`for num in [1, 2, 3, 4, 5]:
 print(num)`

4th iteration: 
`for num in [1, 2, 3, 4, 5]:
 print(num)`

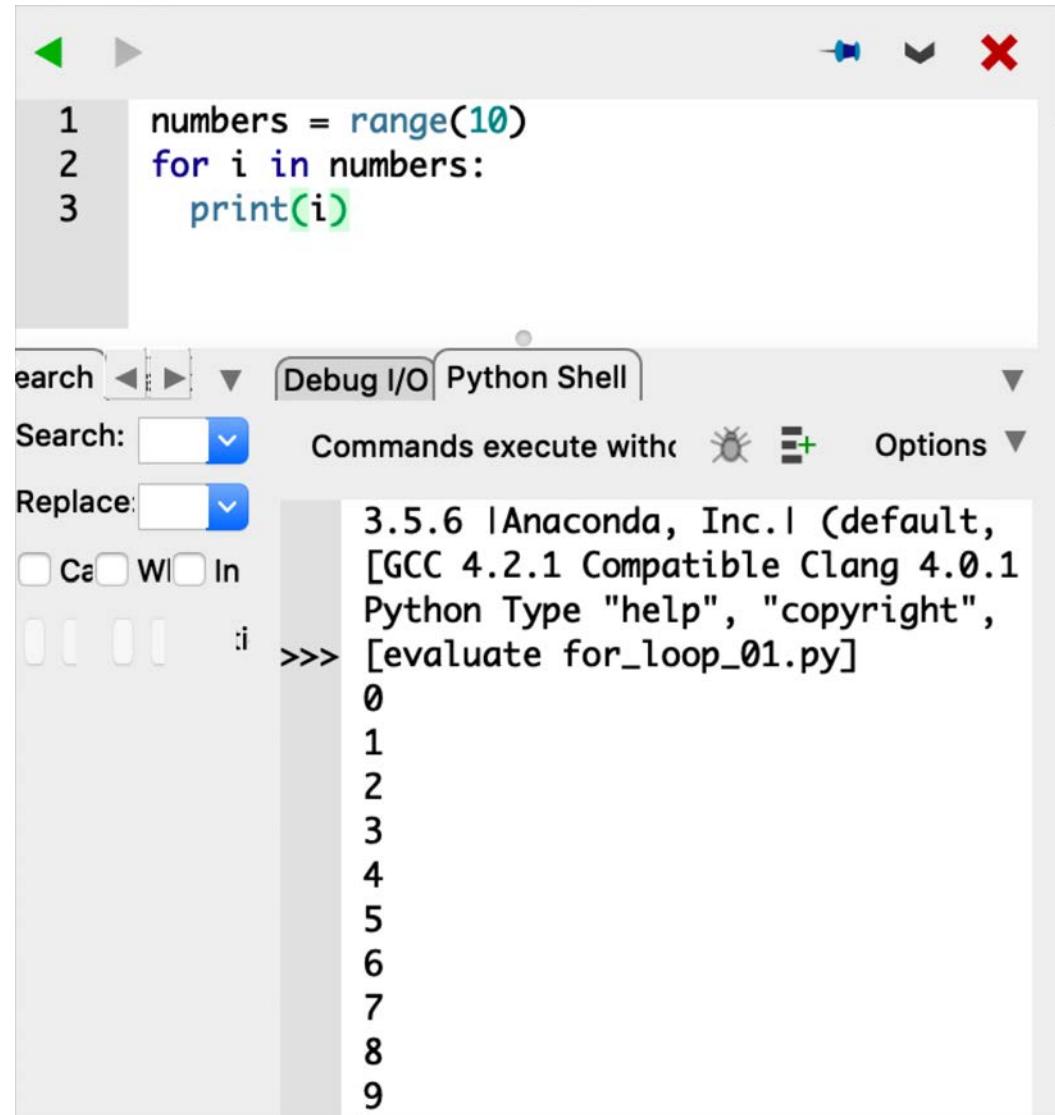
5th iteration: 
`for num in [1, 2, 3, 4, 5]:
 print(num)`

For Loops

Notice how each call of print at each loop will print at a different line.

How do we print numbers 0 to 9 all on the same line (0123456789)?

Hint: print(i,end="")



The screenshot shows a Python IDE interface. At the top, there is a code editor window containing the following Python code:

```
1 numbers = range(10)
2 for i in numbers:
3     print(i)
```

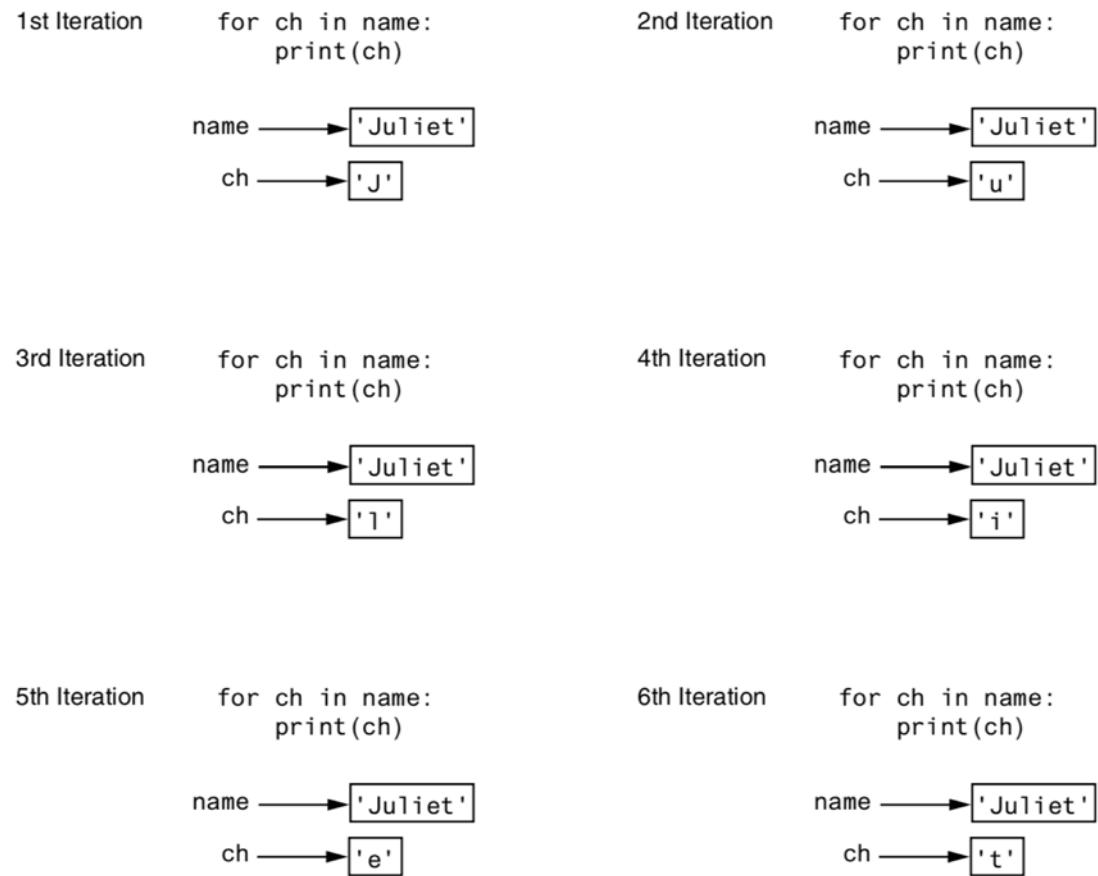
Below the code editor is a toolbar with various icons. The "Python Shell" tab is currently selected. In the Python Shell tab, the following text is displayed:

search < > ▾ Debug I/O Python Shell ▾
Search: ▾ Commands execute with: Options ▾
Replace: ▾
 Ca Wi In
... :>>> 3.5.6 |Anaconda, Inc.| (default,
[GCC 4.2.1 Compatible Clang 4.0.1
Python Type "help", "copyright",
[evaluate for_loop_01.py]
0
1
2
3
4
5
6
7
8
9

The output in the shell shows the numbers 0 through 9 printed on a single line, separated by newlines, demonstrating the use of the `print(i)` statement with the `end=""` parameter.

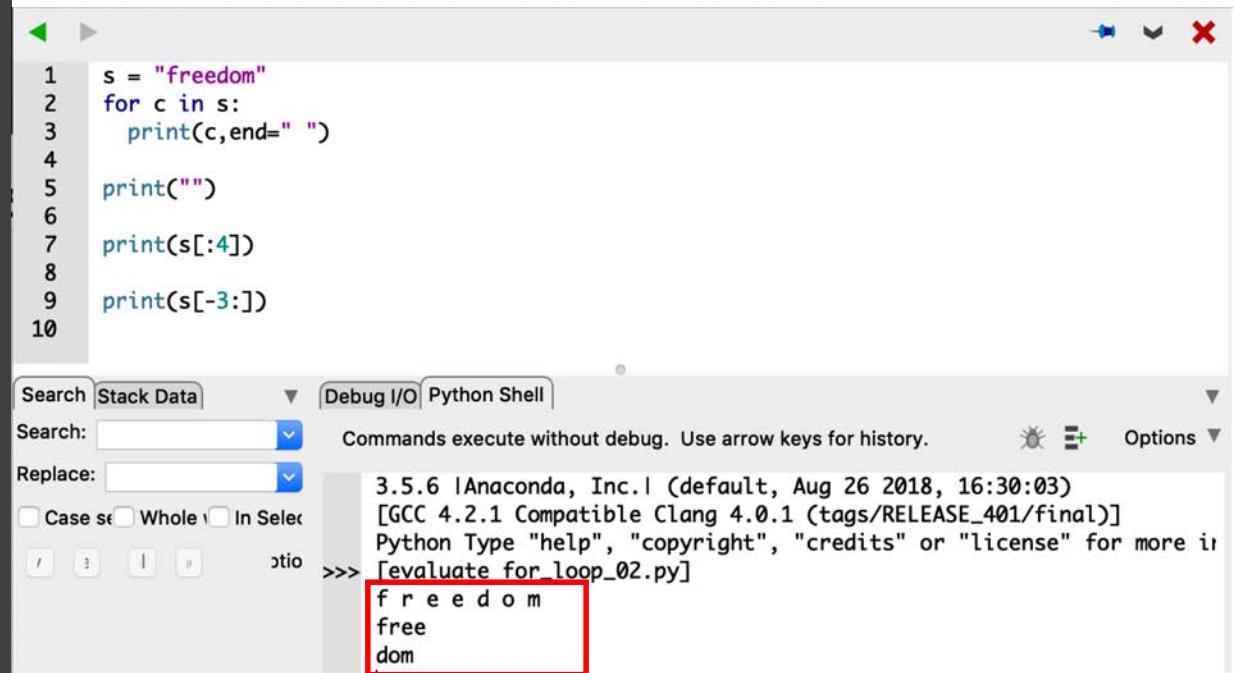
For Loops with string

- A string is a sequence, like a list
- The for loop works similarly with strings



For Loops with string

- Slicing works for any sequence, so it works for strings too.
 - `[4]` gets from the start till the fourth character
 - `[-3:]` gets the last third till the last character.



The screenshot shows a Python code editor and an integrated terminal window. The code in the editor is:

```
1 s = "freedom"
2 for c in s:
3     print(c,end=" ")
4
5 print("")
6
7 print(s[:4])
8
9 print(s[-3:])
```

The terminal window shows the output of the code:

```
3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for more information
>>> [evaluate for_loop_02.py]
f r e e d o m
free
dom
```

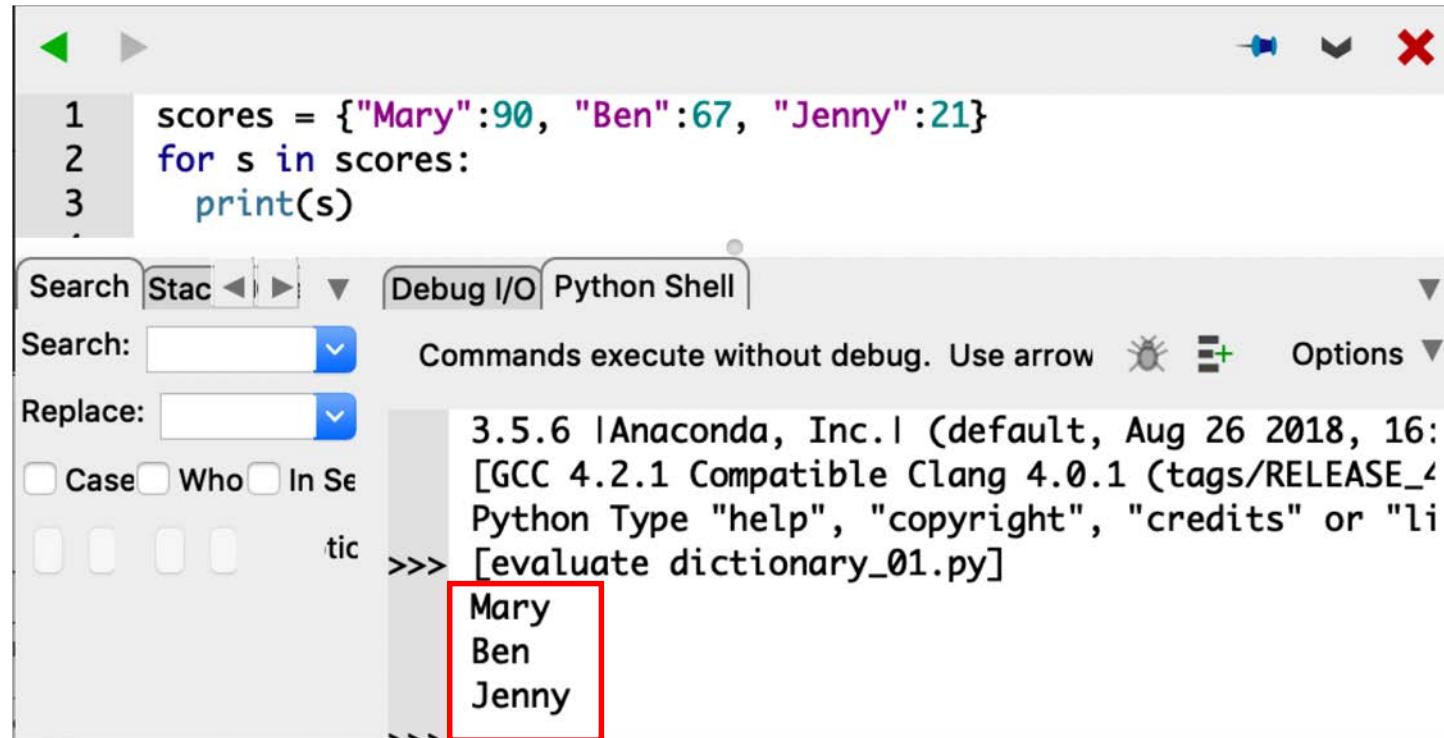
The output lines for "free" and "dom" are highlighted with a red rectangle.

Data Types - Dictionary

- A dictionary stores multiple key-value pairs
- E.g. In the first row of output, the dictionary contains 3 key-value pairs (which are the keys?)
- Every key is unique; no duplicate key within a dictionary
- A dictionary uses a set of curly brackets to store its key-value pairs {...}
=> Contrast with a list that uses square brackets to store its objects [...]
- To access a value in the dictionary, we use the key as an index

```
{'year': '1995', 'type_of_public_transport': 'MRT', 'average_ridership': '740000'}  
{'year': '1995', 'type_of_public_transport': 'LRT', 'average_ridership': '0'}  
{'year': '1995', 'type_of_public_transport': 'Bus', 'average_ridership': '3009000'}  
{'year': '1995', 'type_of_public_transport': 'Taxi', 'average_ridership': '0'}  
{'year': '1996', 'type_of_public_transport': 'MRT', 'average_ridership': '850000'}  
{'year': '1996', 'type_of_public_transport': 'LRT', 'average_ridership': '0'}
```

Data Types - Dictionary



The screenshot shows a Python code editor window with the following code:

```

1 scores = {"Mary":90, "Ben":67, "Jenny":21}
2 for s in scores:
3     print(s)

```

The code is run in a Python Shell tab, which displays the output:

```

3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16: [GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_4 Python Type "help", "copyright", "credits" or "li
tic >>> [evaluate dictionary_01.py]
Mary
Ben
Jenny

```

The output lines 'Mary', 'Ben', and 'Jenny' are highlighted with a red box.

- How does a `for` loop work on dictionaries?
- Doing '`for s in scores`' in the above code will assign the value of each key to `s`
- Change '`print(s)`' to '`print(s, scores[s])`', what do you get?

Exercise

Even Odd Counter

Write and test a program that will read 10 positive integer numbers, determine if it is even or odd, keep count of the number of even and odd numbers and display the final outcome as follows:

Enter number 1: 12

Enter number 2: 7

• • •

Enter number 10 : 67

Even #: 4

Odd #: 6

- Q: What if a user does not enter a positive integer?



Hint: use `list.append()` if necessary

Functions

A function is a group of statements that exist within a program for the purpose of performing a specific task.

This program is one long, complex sequence of statements.

statement
statement

In this program the task has been divided into smaller tasks, each of which is performed by a separate function.

```
def function1():  
    statement      function  
    statement  
    statement  
    statement
```

```
def function2():  
    statement      function  
    statement  
    statement  
    statement
```

```
def function3():  
    statement      function  
    statement  
    statement  
    statement
```

```
def function4():  
    statement      function  
    statement  
    statement  
    statement
```

Functions

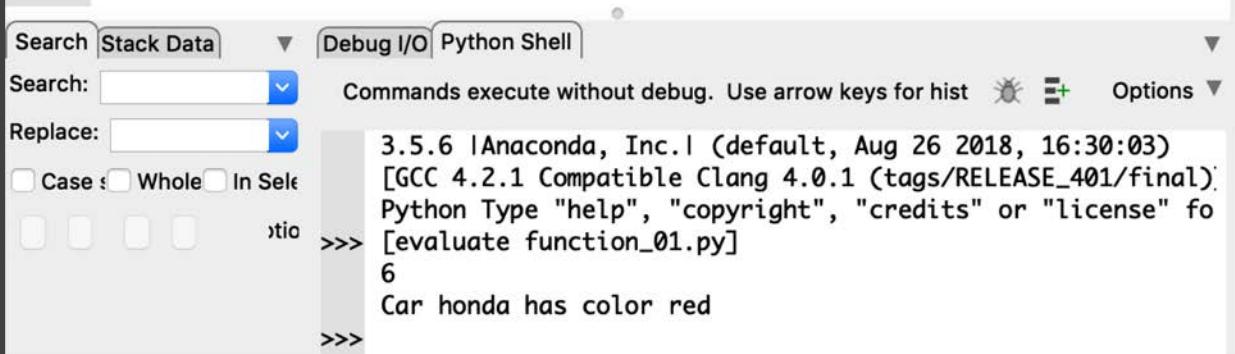
Passing parameters

- Arguments must be passed in order

Alternatively, use parameter names to identify the arguments

Be sure to have
indentation

```
identCar
1 def multiply(a,b):
2     return a*b
3
4 print(multiply(2,3))
5
6 def identCar(car,colour):
7     return "Car %s has color %s" % (car, colour)
8
9 print(identCar(colour="red", car="honda"))
10
```

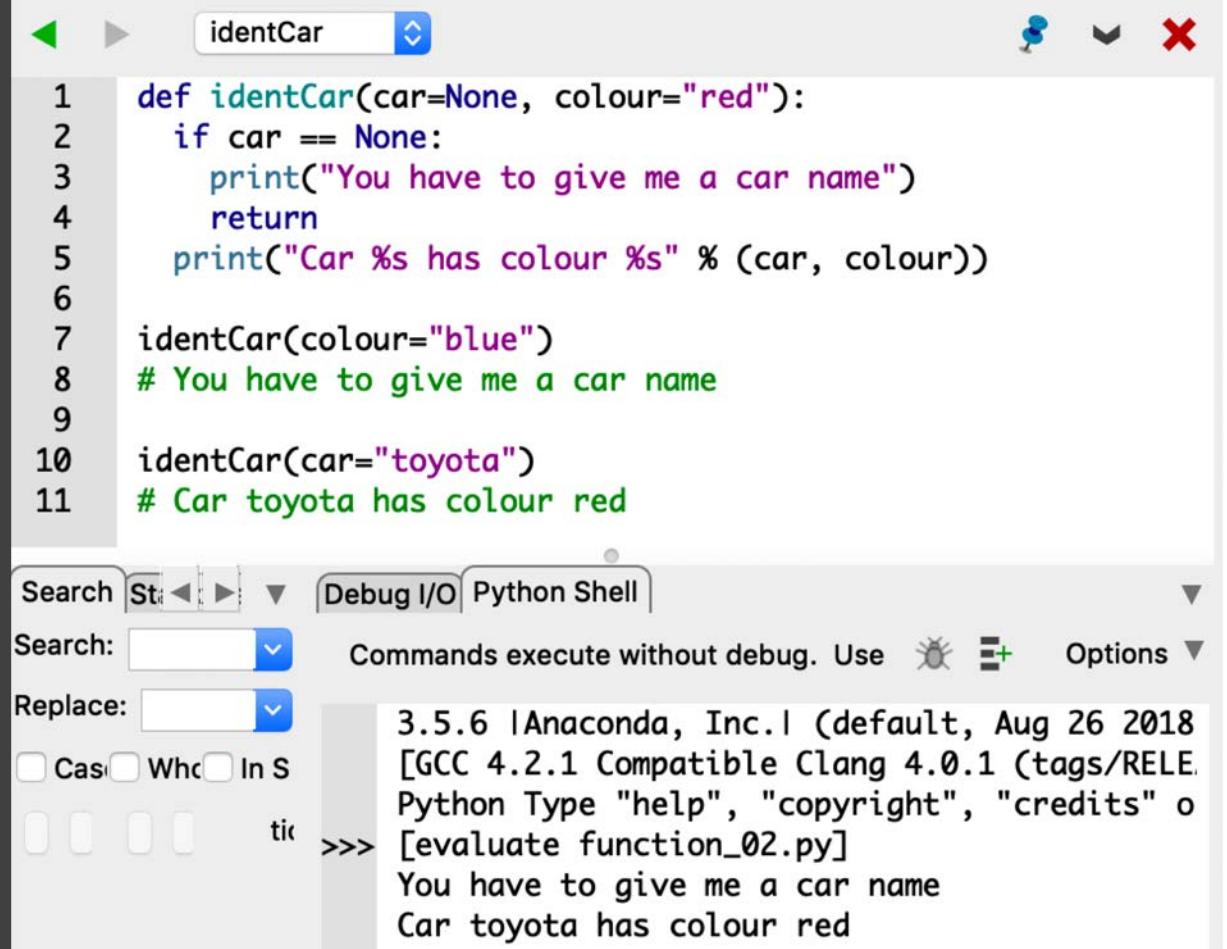


The screenshot shows a Python IDE interface with a code editor and a Python Shell tab. The code editor contains the same Python script as above. The Python Shell tab shows the following output:

```
Search Stack Data Debug I/O Python Shell
Search: Replace: Commands execute without debug. Use arrow keys for hist Options
Case Whole In Selection
3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for
[evaluate function_01.py]
6
Car honda has color red
```

Functions - Default Parameters

Default parameters values
and checking if parameter
has been passed



The screenshot shows a Python code editor window with a script named `identCar`. The code defines a function `identCar` that takes two parameters: `car` (with a default value of `None`) and `colour` (with a default value of `"red"`). The function checks if `car` is `None`, prints a message if it is, and then prints the car's name and colour. The code then demonstrates calling the function with different arguments.

```
def identCar(car=None, colour="red"):
    if car == None:
        print("You have to give me a car name")
        return
    print("Car %s has colour %s" % (car, colour))

identCar(colour="blue")
# You have to give me a car name

identCar(car="toyota")
# Car toyota has colour red
```

Below the code editor is a Python Shell interface. It shows the environment details (Anaconda 3.5.6, Python 2.7.14) and the command-line interaction. The user runs the script and gets the expected output: a prompt for a car name followed by the printed car name and its colour.

Search **St** Debug I/O Python Shell

Search: Commands execute without debug. Use  Options 

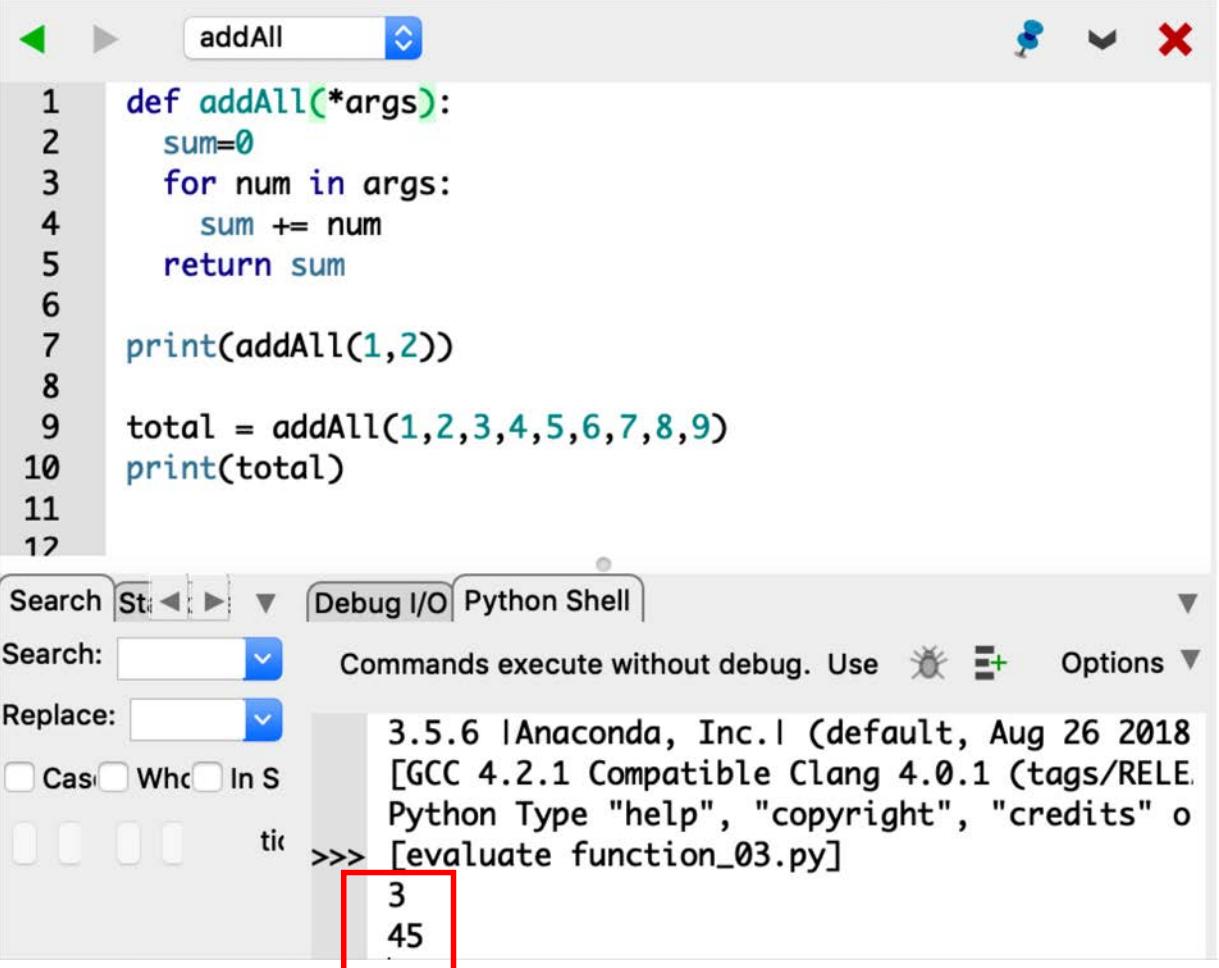
Replace:   

`>>> [evaluate function_02.py]`

You have to give me a car name
Car toyota has colour red

Functions - Arbitrary argument list

If you don't know how many parameters the function will receive, you can use *args, which will be a list.



```
1 def addAll(*args):
2     sum=0
3     for num in args:
4         sum += num
5     return sum
6
7 print(addAll(1,2))
8
9 total = addAll(1,2,3,4,5,6,7,8,9)
10 print(total)
11
12
```

The screenshot shows a Python code editor with the file named 'addAll'. The code defines a function 'addAll' that takes a variable number of arguments (*args) and returns their sum. It also includes two print statements: one for the sum of 1 and 2, and another for the sum of all integers from 1 to 9. Below the code editor is a Python Shell window. The shell shows the command 'evaluate function_03.py' being run, followed by the output '3' and '45'. The number '3' is highlighted with a red box.

Exercise

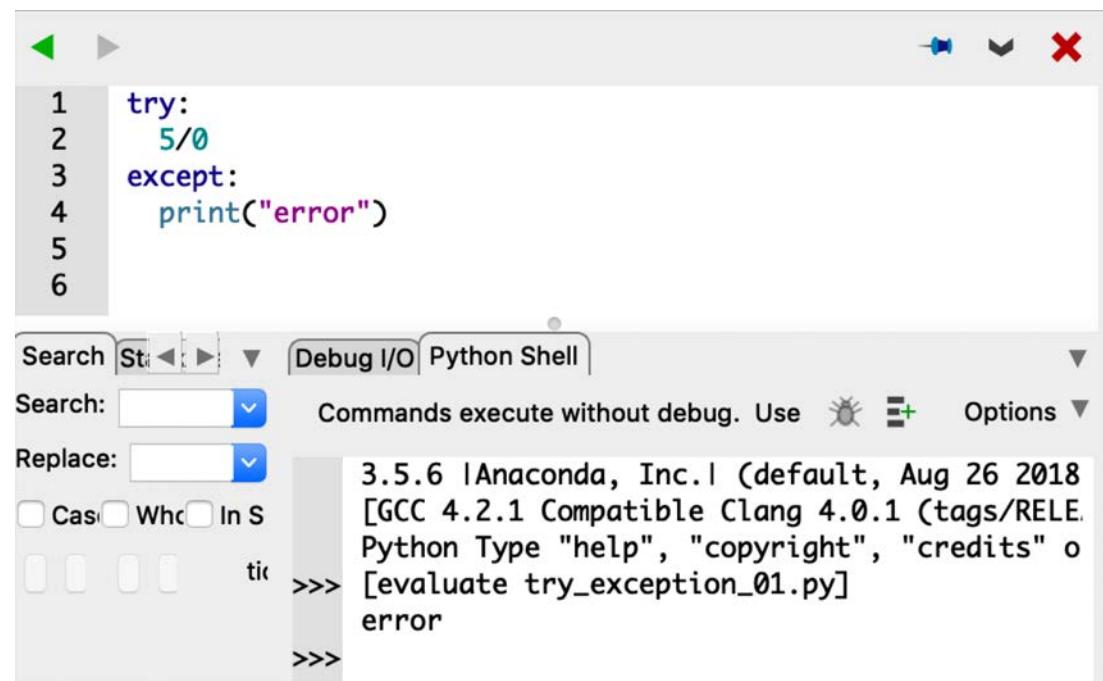
Arbitrary argument list

Create a function that takes in an unknown amount of parameters and returns the sum.



try .. except

- Error handling is done through the use of exceptions that are caught in try blocks and handled in except blocks



```

1 try:
2     5/0
3 except:
4     print("error")
5
6

```

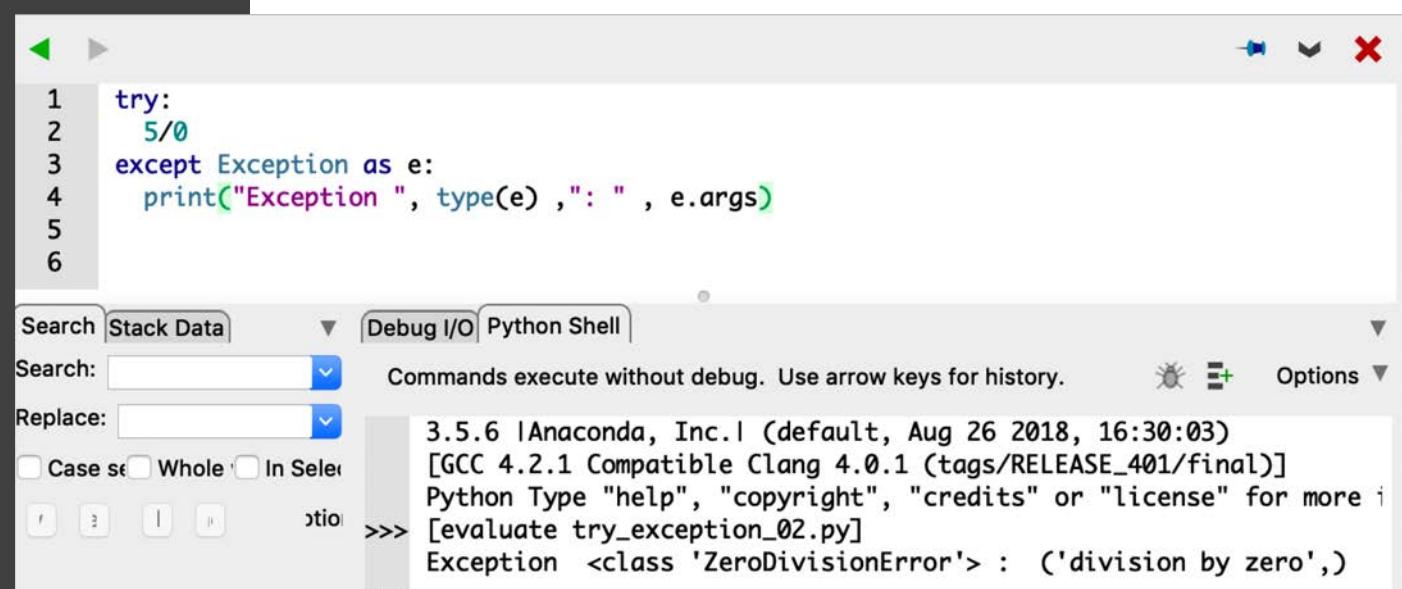
Search **Stack Data** Debug I/O Python Shell

Search: Commands execute without debug. Use Options

Replace:

< Case > Whole < In Selection >

>>> 3.5.6 |Anaconda, Inc.| (default, Aug 26 2018
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for more information
>>> [evaluate try_exception_01.py]
error
>>>



```

1 try:
2     5/0
3 except Exception as e:
4     print("Exception ", type(e) ,": " , e.args)
5
6

```

Search **Stack Data** Debug I/O Python Shell

Search: Commands execute without debug. Use arrow keys for history.

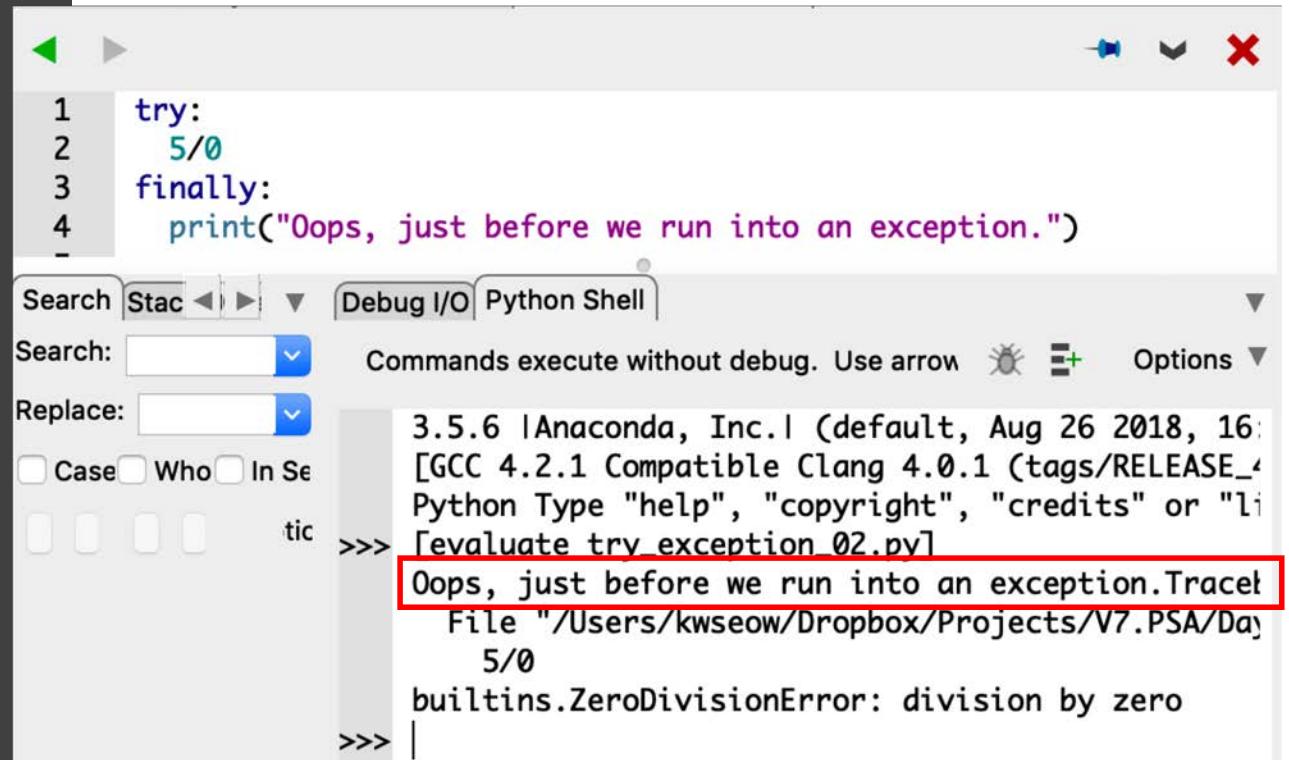
Replace:

< Case > Whole < In Selection >

>>> 3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for more information
>>> [evaluate try_exception_02.py]
Exception <class 'ZeroDivisionError'> : ('division by zero',)

try .. except

- You can also use the finally block. The code in the finally block will be executed regardless of whether an exception occurs.



The screenshot shows a Python terminal window with the following code:

```
1 try:
2     5/0
3 finally:
4     print("Oops, just before we run into an exception.")
```

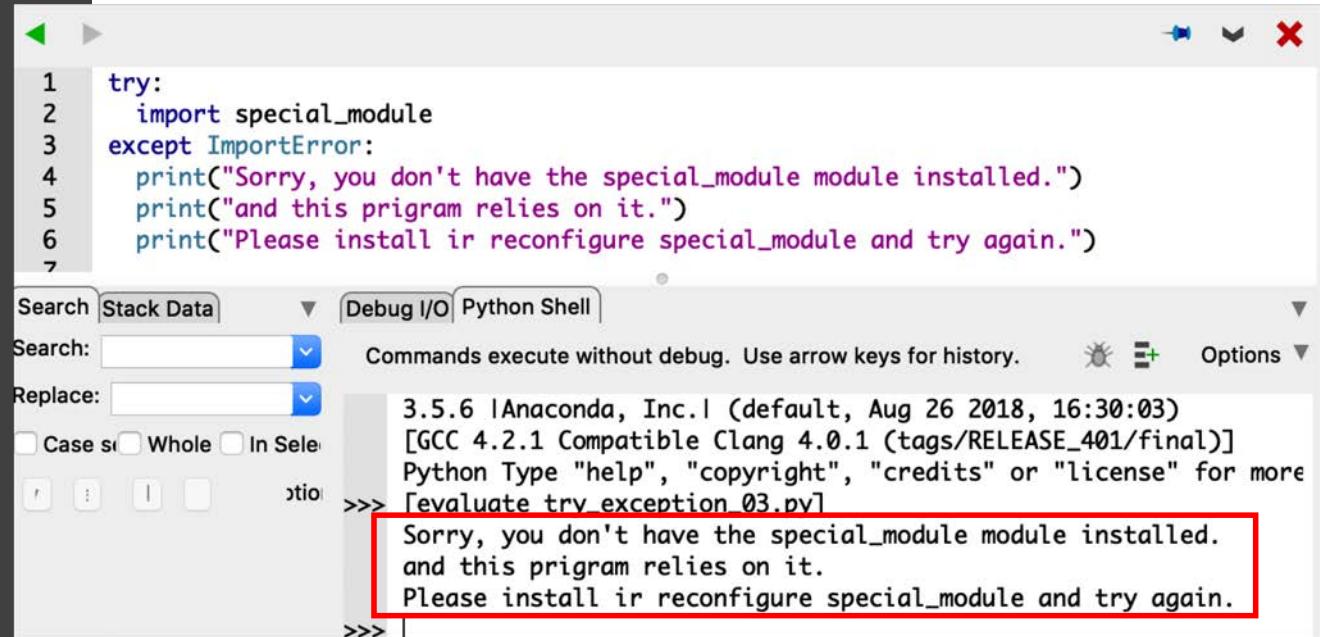
The terminal output is:

```
>>> [evaluate try_exception_02.py]
Oops, just before we run into an exception.
Traceback (most recent call last):
File "/Users/kwseow/Dropbox/Projects/V7.PSA/Day 1/try_except_02.py", line 2, in <module>
    5/0
builtins.ZeroDivisionError: division by zero
>>> |
```

The output message "Oops, just before we run into an exception." is highlighted with a red box.

try .. except

- A good use for try expect is to check if the user has the specific library installed and if not, explains to the user what to do:



The screenshot shows a Python code editor with the following script content:

```
1 try:
2     import special_module
3 except ImportError:
4     print("Sorry, you don't have the special_module module installed.")
5     print("and this program relies on it.")
6     print("Please install or reconfigure special_module and try again.")
```

The code is run in a Python Shell, and the output is:

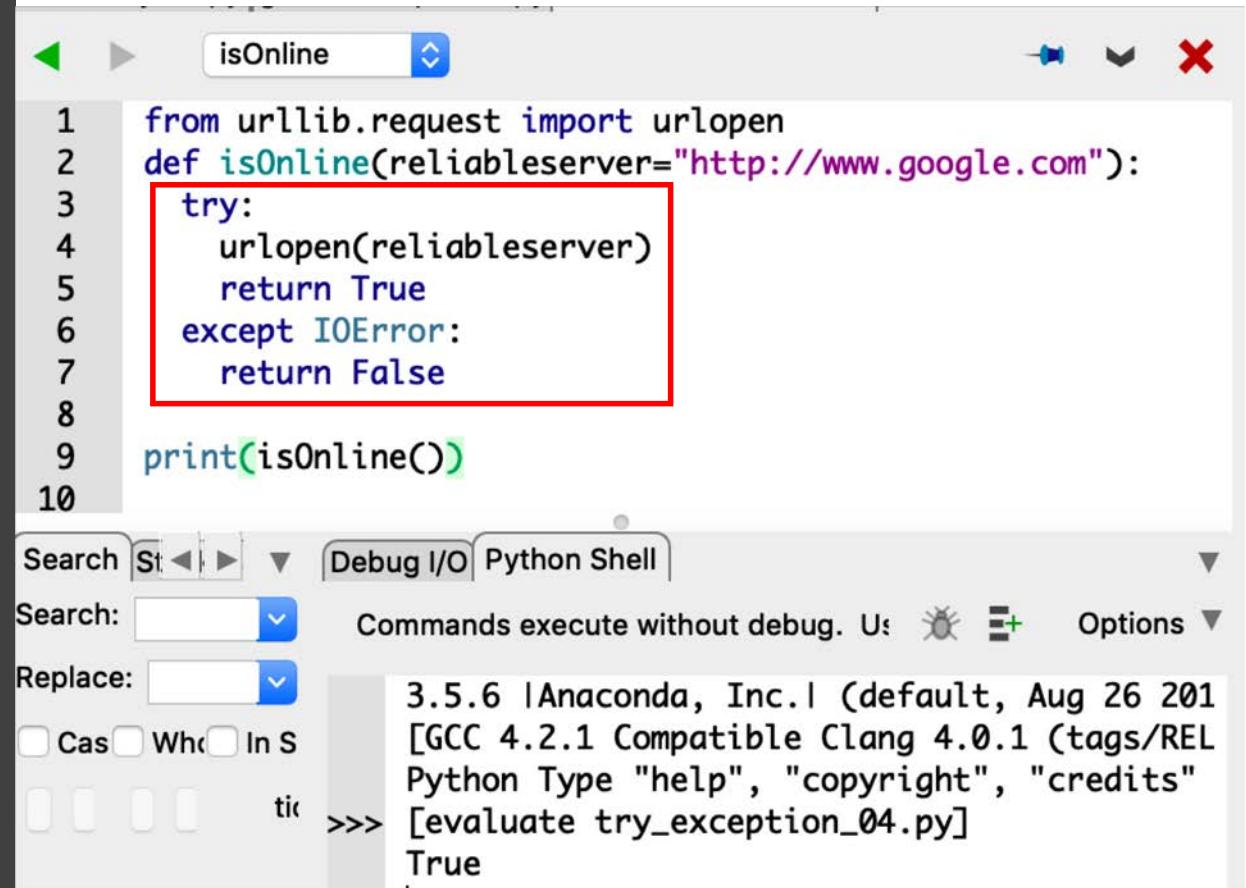
```
>>> Evaluate try_exception_03.py
3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 16:30:03)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for more
information.

Sorry, you don't have the special_module module installed.
and this program relies on it.
Please install or reconfigure special_module and try again.
```

The last three lines of the output are highlighted with a red box.

try .. except

- Another example is to check if a website is available:



```
from urllib.request import urlopen
def isOnline(reliableserver="http://www.google.com"):
    try:
        urlopen(reliableserver)
        return True
    except IOError:
        return False

print(isOnline())
```

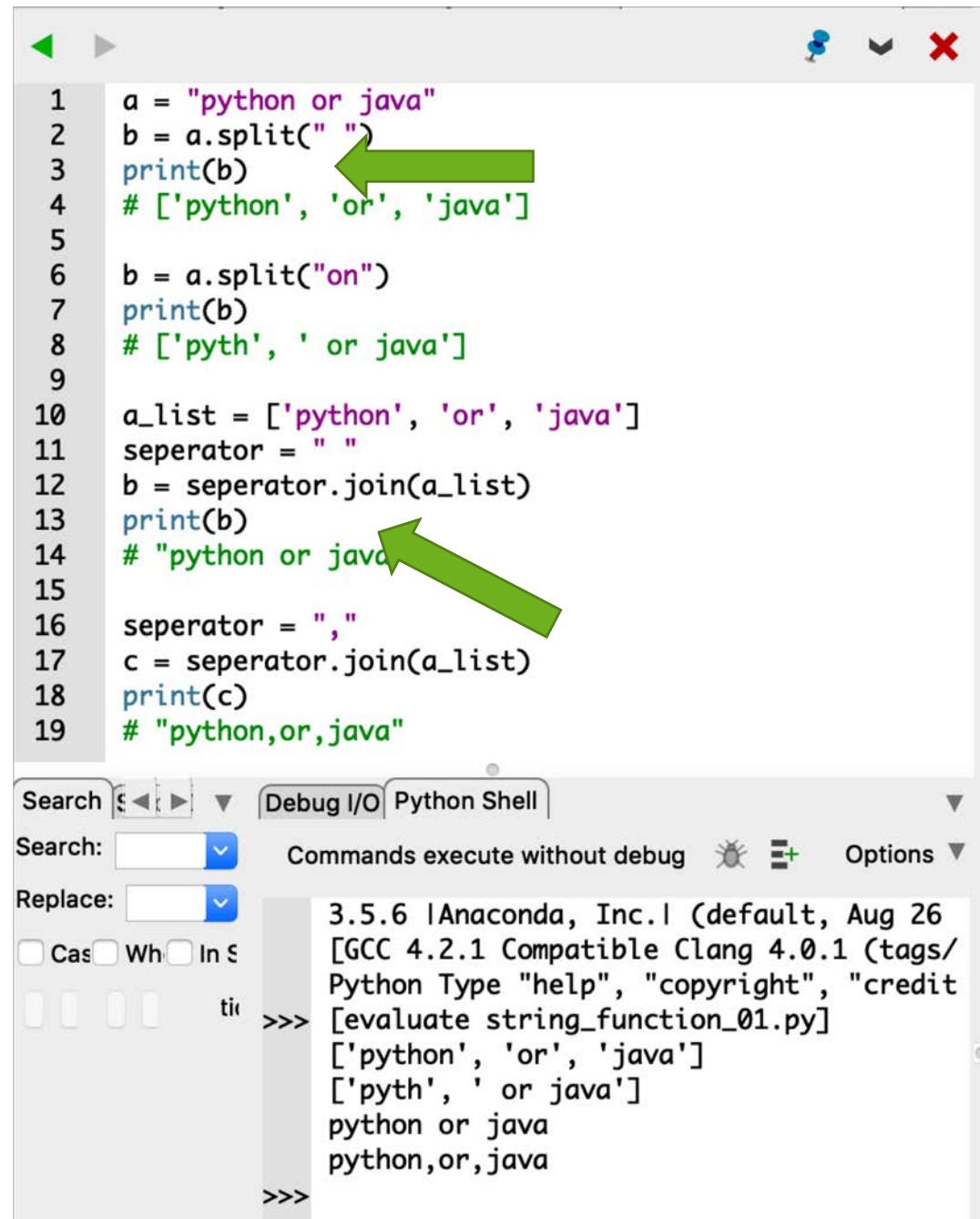
The code above defines a function `isOnline` that attempts to open a URL using `urlopen`. If it succeeds, it returns `True`; if it fails (due to an `IOError`), it returns `False`. The `try..except` block is highlighted with a red rectangle.

Below the code, the Python Shell tab of the IDE shows the output of running the script:

```
3.5.6 |Anaconda, Inc.| (default, Aug 26 201
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/REL
Python Type "help", "copyright", "credits"
[evaluate try_exception_04.py]
True
```

String functions

- `split()` - returns a list of strings after breaking the given string by the specified separator.
- `Join()` - returns a string in which the string elements of sequence have been joined by str separator.



```

1 a = "python or java"
2 b = a.split(" ")
3 print(b)
4 # ['python', 'or', 'java']
5
6 b = a.split("on")
7 print(b)
8 # ['pyth', ' or java']
9
10 a_list = ['python', 'or', 'java']
11 seperator = " "
12 b = seperator.join(a_list)
13 print(b)
14 # "python or java"
15
16 seperator = ","
17 c = seperator.join(a_list)
18 print(c)
19 # "python,or,java"

```

The screenshot shows a Python code editor with the following code:

```

1 a = "python or java"
2 b = a.split(" ")
3 print(b)
4 # ['python', 'or', 'java']
5
6 b = a.split("on")
7 print(b)
8 # ['pyth', ' or java']
9
10 a_list = ['python', 'or', 'java']
11 seperator = " "
12 b = seperator.join(a_list)
13 print(b)
14 # "python or java"
15
16 seperator = ","
17 c = seperator.join(a_list)
18 print(c)
19 # "python,or,java"

```

Two green arrows highlight the output of the print statements at lines 3 and 13. The output is visible in the Python Shell tab below:

```

Search: Debug I/O Python Shell
Search: Replace: Commands execute without debug Options
Cas Wh In S
>>> 3.5.6 |Anaconda, Inc.| (default, Aug 26
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/
Python Type "help", "copyright", "credit
[evaluate string_function_01.py]
['python', 'or', 'java']
['pyth', ' or java']
python or java
python,or,java
>>>

```

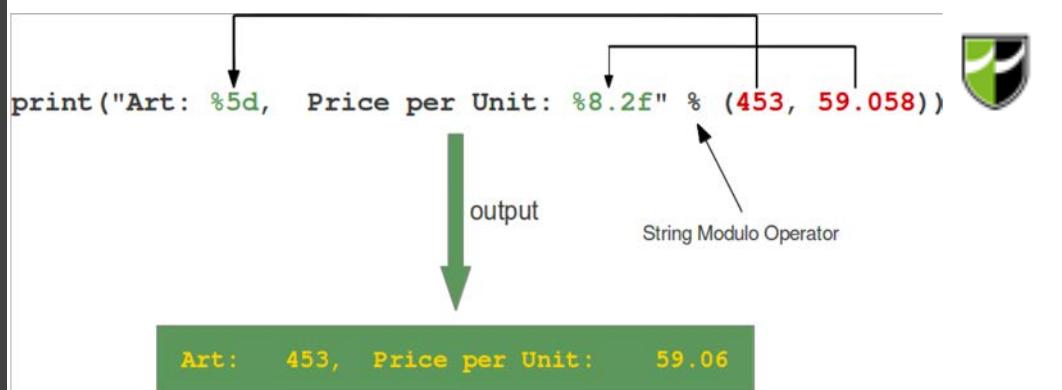
Exercise

Find Longest Word

- Create the function `findLongestWord` that takes in a sentence and returns the longest word. Hint: Use `split()`



String formatting



A screenshot of a Python IDE showing code execution. The code is:

```
1 import math
2 a = math.pi
3 print(a)
4 # 3.141592653589793
5
6 b = 5
7 c = "python"
8 line = "%s %f %d" % (c, a, b)
9 print(line)
10 # python 3.141593 5
11
12 line = "%03d" % (b)
13 print(line)
14 # 005
```

The interface includes tabs for "Search", "Debug I/O", and "Python Shell". The "Python Shell" tab shows the output of the code execution:

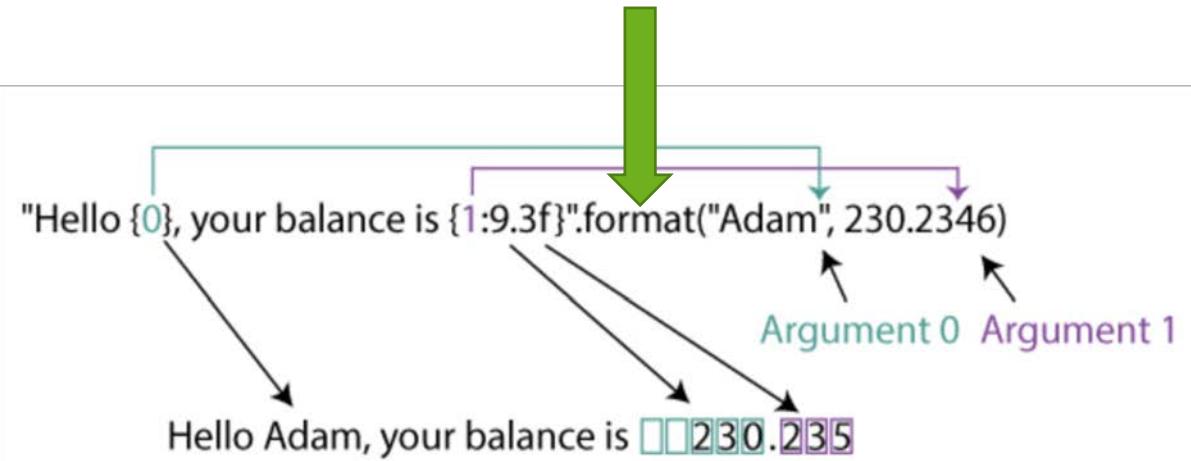
```
3.141592653589793
python 3.141593 5
005
```

Additional reading:

<https://pyformat.info/>

<https://docs.python.org/3/library/stdtypes.html#string-formatting-operations>

String format()



Additional reading:

<https://www.programiz.com/python-programming/methods/string/format>

Exercise – Xmas Tree

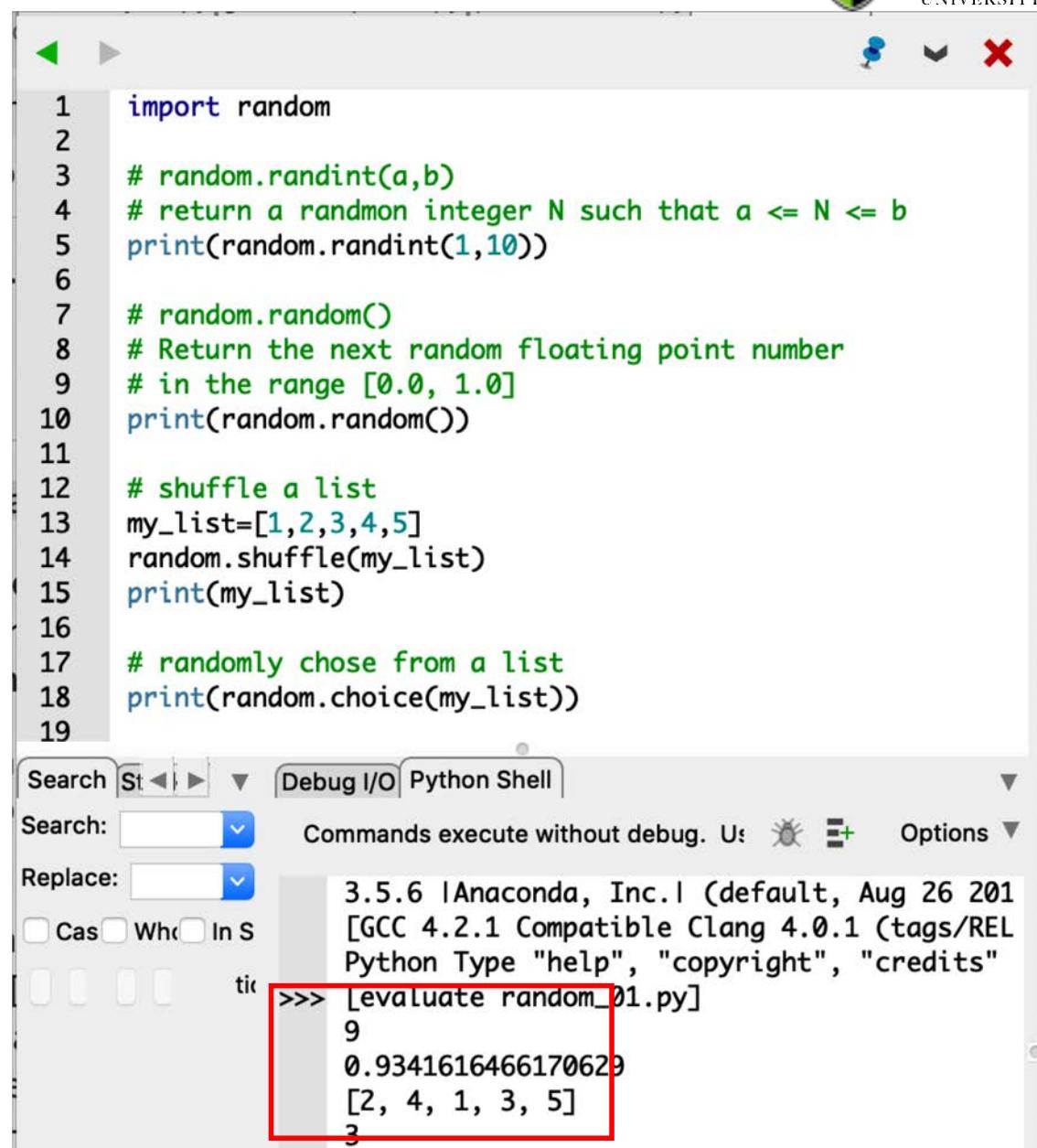
- Question: Using string formatting and a loop, try to print the following xmas tree:

```
#  
###  
####  
#####  
######  
#######  
########
```



The random library

implements pseudo-random number generators for various distributions.



A screenshot of a Python code editor showing a script named `random_01.py`. The code demonstrates various functions from the `random` module:

```
1 import random
2
3 # random.randint(a,b)
4 # return a random integer N such that a <= N <= b
5 print(random.randint(1,10))
6
7 # random.random()
8 # Return the next random floating point number
9 # in the range [0.0, 1.0]
10 print(random.random())
11
12 # shuffle a list
13 my_list=[1,2,3,4,5]
14 random.shuffle(my_list)
15 print(my_list)
16
17 # randomly chose from a list
18 print(random.choice(my_list))
19
```

The output window shows the results of running the script:

```
>>> [evaluate random_01.py]
9
0.9341616466170629
[2, 4, 1, 3, 5]
3
```

Additional reading:
<https://docs.python.org/3/library/random.html>

Exercise - Guessing Game

- Create a random number between 1 and 20 and prompt the user to guess the secret number. He is allowed a maximum of 6 guesses after which the secret number will be displayed and the program exits. For every guess, the program will display a message saying if the number guessed is higher or lower than the secret number. If he guessed the correct number, the program will display the number of tries he had taken and the program exits.



Exercise - Guessing Game

- Sample output

```
What is your name?  
John  
Well, John, I am thinking of a number between 1 and 20  
Take a guess  
5  
Your guess is too low.  
Take a guess  
10  
Your guess is too low.  
Take a guess  
15  
Your guess is too high.  
Take a guess  
12  
Your guess is too low.  
Take a guess  
14  
Good job, John! You guessed my number in 5 guesses!  
  
Process finished with exit code 0
```

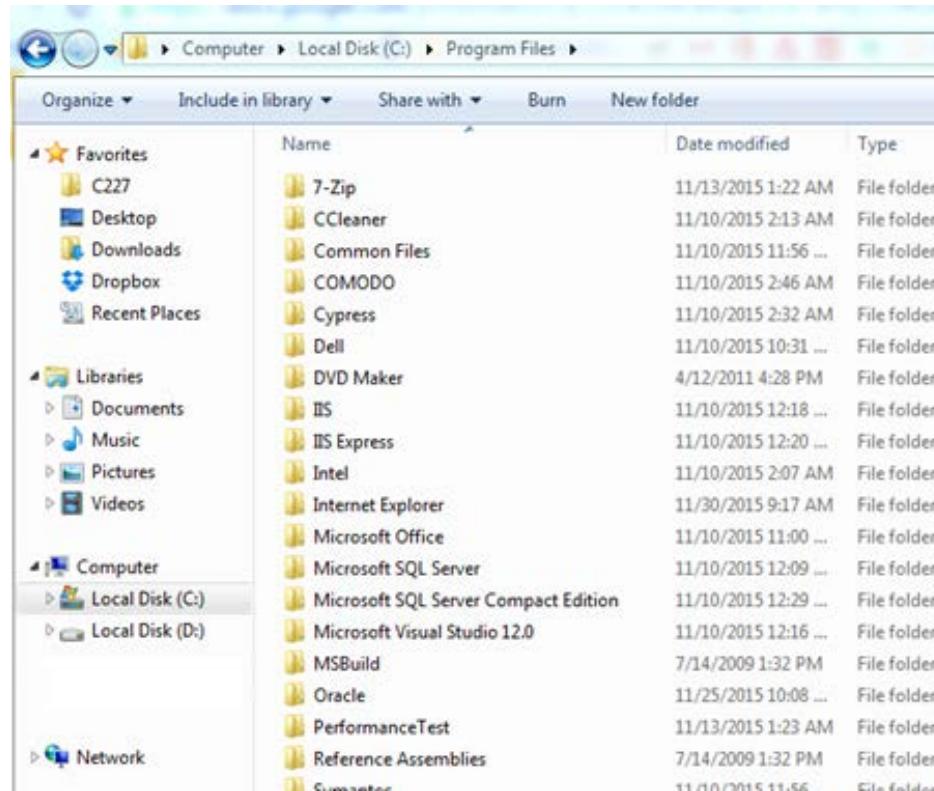


```
What is your name?  
John  
Well, John, I am thinking of a number between 1 and 20  
Take a guess  
10  
Your guess is too high.  
Take a guess  
10  
Your guess is too high.  
Take a guess  
10  
Your guess is too high.  
Take a guess  
10  
Your guess is too high.  
Take a guess  
10  
Your guess is too high.  
nope. The number I was thinking of was 6  
  
Process finished with exit code 0
```

Search for file by filename

Let's say you want to find a file by filename "readme.txt" and it's somewhere in

C:\Program Files\



Name	Date modified	Type
7-Zip	11/13/2015 1:22 AM	File folder
CCleaner	11/10/2015 2:13 AM	File folder
Common Files	11/10/2015 11:56 ...	File folder
COMODO	11/10/2015 2:46 AM	File folder
Cypress	11/10/2015 2:32 AM	File folder
Dell	11/10/2015 10:31 ...	File folder
DVD Maker	4/12/2011 4:28 PM	File folder
IIS	11/10/2015 12:18 ...	File folder
IIS Express	11/10/2015 12:20 ...	File folder
Intel	11/10/2015 2:07 AM	File folder
Internet Explorer	11/30/2015 9:17 AM	File folder
Microsoft Office	11/10/2015 11:00 ...	File folder
Microsoft SQL Server	11/10/2015 12:09 ...	File folder
Microsoft SQL Server Compact Edition	11/10/2015 12:29 ...	File folder
Microsoft Visual Studio 12.0	11/10/2015 12:16 ...	File folder
MSBuild	7/14/2009 1:32 PM	File folder
Oracle	11/25/2015 10:08 ...	File folder
PerformanceTest	11/13/2015 1:23 AM	File folder
Reference Assemblies	7/14/2009 1:32 PM	File folder
Windows	11/10/2015 11:56 ...	File folder

Search for file by filename

We need to work with the os library, so we import it at the start.

The folder we search in is stored in the variable where.

At line 6 we create a new function called searchByName.
It has one parameter, which is name and contains the name of the file we are looking for.

In this function we walk through all directories and files.
Then for each file we compare if it is the filename we are looking for.

```
1 import os
2
3 where = "C:\\\\Program Files\\\\"
4 #where = "C:\\\\Program Files (x86) \\\\"
5
6 def searchByName(name):
7     for root, dirs, files in os.walk(where):
8         for file in files:
9             if file == name:
10                 print(os.path.join(root, file))
11
12 searchByName ("readme.txt")
13
```

Search for file by filename

Let's say you want to know the combined filesize of all these readme.txt files.

Initialize a new variable totalSize to 0 at the start of our function.

Then for each readme.txt we add the size to this variable with

```
totalSize +=  
os.path.getsize(os.path.join(root,file  
))
```

Then we make this function return this value with return totalSize at the very end of the function with
return totalSize

```
def searchByName(name):  
    totalSize = 0  
    for root, dirs, files in os.walk(where):  
        for file in files:  
            if file == name:  
                print(os.path.join(root,file))  
                totalSize += os.path.getsize(os.path.join(root,file))  
    return totalSize
```

Make the amendments as indicated above, then run it and observe.

Did it show you the total ?

Search for file by filename

The function returned the value, but we didn't do anything with it.

When we call the function, we can assign the value to a variable.
Then we can print the content.

```
total = searchByName("readme.txt")
print ("Total is : %d" % (total))
print ("All done")
```

```
C:\Program Files\Unity\MonoDevelop>Addins\MonoDevelop.AspNet\Schemas\readme.txt
C:\Program Files\Unity\MonoDevelop>Addins\MonoDevelop.XmlEditor\schemas\readme.txt
Total is 17569
All done.
>>>
```

Exercise – listing files

Can you update the program to show the individual file size of all the files?

```
>>>
1761 C:\Program Files\7-Zip\readme.txt
 83 C:\Program Files\Unity\Editor\Data\Playba
 62 C:\Program Files\Unity\Editor\Data\Playba
549 C:\Program Files\Unity\Editor\Data\Playba
695 C:\Program Files\Unity\Editor\Data\Playba
126 C:\Program Files\Unity\Editor\Data\Playba
 89 C:\Program Files\Unity\Editor\Data\Playba
 66 C:\Program Files\Unity\Editor\Data\Playba
250 C:\Program Files\Unity\Editor\Data\Playba
 25 C:\Program Files\Unity\Editor\Data\Playba
11333 C:\Program Files\Unity\Editor\Data\Playba
 718 C:\Program Files\Unity\Editor\Data\Playba
 906 C:\Program Files\Unity\MonoDevelop\Addin:
 906 C:\Program Files\Unity\MonoDevelop\Addin:
Total is 17569
All done.
>>>
```



Search for file by filename

Since we use the full filename multiple times, it makes sense to store it in a separate variable.
Same for the filesize.

The %6d makes sure the output looks nice, in columns format.

```
def searchByName(name):
    totalSize = 0
    for root, dirs, files in os.walk(where):
        for file in files:
            if file == name:
                fullName = os.path.join(root, file)
                fileSize = os.path.getsize(fullName)
                print("%6d %s"%(fileSize, fullName))
                totalSize += fileSize
    return totalSize
```

File files larger than xxMB

Create a new function called `searchBySize`, that takes one parameter and only prints those files larger than that parameter.

for example:
`searchBySize(50000000)`
will only print the full path and filename of those files that exceed 50Mb

You still need the if statement but it's not based on the name.
Do you know the `fileSize` at the point of the if statement ?

```
def searchByName(name):
    totalSize = 0
    for root, dirs, files in os.walk(where):
        for file in files:
            if file == name:
                fullName = os.path.join(root, file)
                fileSize = os.path.getsize(fullName)
                print("%6d %s"%(fileSize, fullName))
                totalSize += fileSize
    return totalSize
```

File files larger than xxMB

Most of the function can be copied, but you need to move the declaration of fullName and fileSize before the if statement.

The if statement then can use the fileSize.

You can still return the totalSize although that was not required.

```
def searchBySize(size):
    totalSize = 0
    for root, dirs, files in os.walk(where):
        for file in files:
            fullName = os.path.join(root, file)
            fileSize = os.path.getsize(fullName)
            if fileSize > size:
                print("%6d %s"%(fileSize, fullName))
                totalSize += fileSize
    return totalSize
```

File files of certain file type

What if you want to find all the files of specific file type ?

Create another function (copy the last) and call it searchByExtension. It has to take one parameter which is the extension to look for.

We can use the buildin function .endswith(“.doc”) in our if statement to compare:
if file.endswith(“.doc”):

You should be able to call this function like:
searchByExtension(“.doc”)

```
def searchByExtension(ext):
    totalSize = 0
    for root, dirs, files in os.walk(where):
        for file in files:
            fullName = os.path.join(root, file)
            fileSize = os.path.getsize(fullName)
            if file.endswith(ext):
                print("%6d %s"%(fileSize, fullName))
                totalSize += fileSize
    return totalSize
```

Again the core is the same, but the if statement is different.

There are other ways to solve this, but this is quite straight forward

Find in files of certain file type

Copy the last function and call it searchByContent.

This new function has to take one more parameter (call it keyword) which is the string.

This string has to be present in the file in order to be counted.

You can combine two conditions in an if statement with **and** :

```
if .... and .... :  
    print "okay"
```

```
def searchByContent(ext, keyword):  
    totalSize = 0  
    for root, dirs, files in os.walk(where):  
        for file in files:  
            fullName = os.path.join(root, file)  
            fileSize = os.path.getsize(fullName)  
            if file.endswith(ext) and keyword in open(fullName, encoding="Latin-1").read():  
                print("%6d %s"%(fileSize, fullName))  
                totalSize += fileSize  
    return totalSize
```

You should be able to use it like this:
`searchByContent(".txt","Copyright")`

To search in text files you can use
`if keyword in open(fullName).read():`

Find in files of certain file type

And this is how it looks like.
This function is slower because it has to go through all the content in the files.

open(fullName).read() literally represents the whole file!
Large files might be a problem because they can not fit in memory at one time.

A better solution would be to load blocks of content, but we save that for another course.

```
def searchByContent(ext, keyword):
    totalSize = 0
    for root, dirs, files in os.walk(where):
        for file in files:
            fullName = os.path.join(root, file)
            fileSize = os.path.getsize(fullName)
            if file.endswith(ext) and keyword in open(fullName, encoding="Latin-1").read():
                print("%6d %s"%(fileSize, fullName))
                totalSize += fileSize
    return totalSize
```

Note:

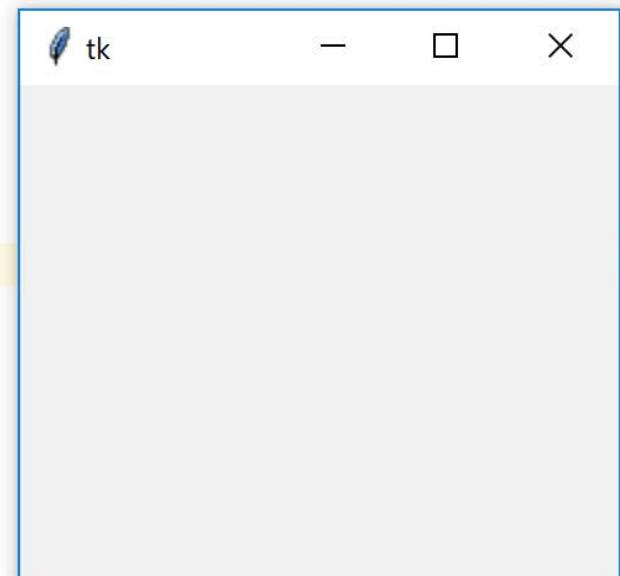
In your if statement, put the file.endswith(ext) first.
This way it will not execute the second part if the first part is already false and thus save precious time.

Graphical User Interface

<https://wiki.python.org/moin/GuiProgramming>

Tkinter – Python's standard GUI library
It is a commonly used GuiProgramming toolkit for Python.

```
1 import tkinter  
2  
3 window = tkinter.Tk()  
4 window.mainloop()  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15
```



Graphical User Interface

Add a button

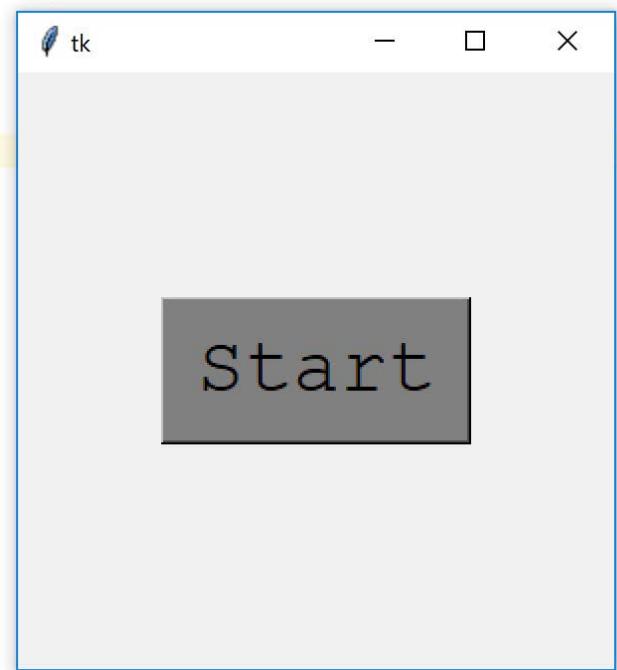
```
1 import tkinter  
2  
3 window = tkinter.Tk()  
4  
5 #Add a button  
6 button = tkinter.Button(window, text="Start")  
7 button.pack()  
8  
9 window.mainloop()  
10  
11
```



Graphical User Interface

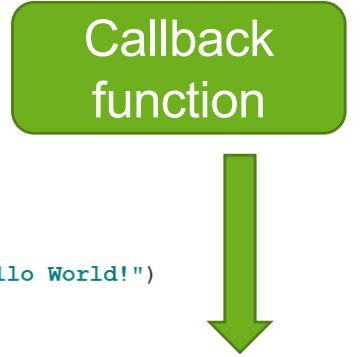
- Set the window's size.
- Configure the colour and position of the button.

```
1 import tkinter
2
3 window = tkinter.Tk()
4
5 #set the window's size
6 window.geometry("300x300")
7
8 #Add a button
9 button = tkinter.Button(window, text="Start", bg="gray")
10 button.config(font=("Courier", 30))
11 button.pack(side="top", expand=tkinter.YES)
12
13 window.mainloop()
14
15
16
17
```



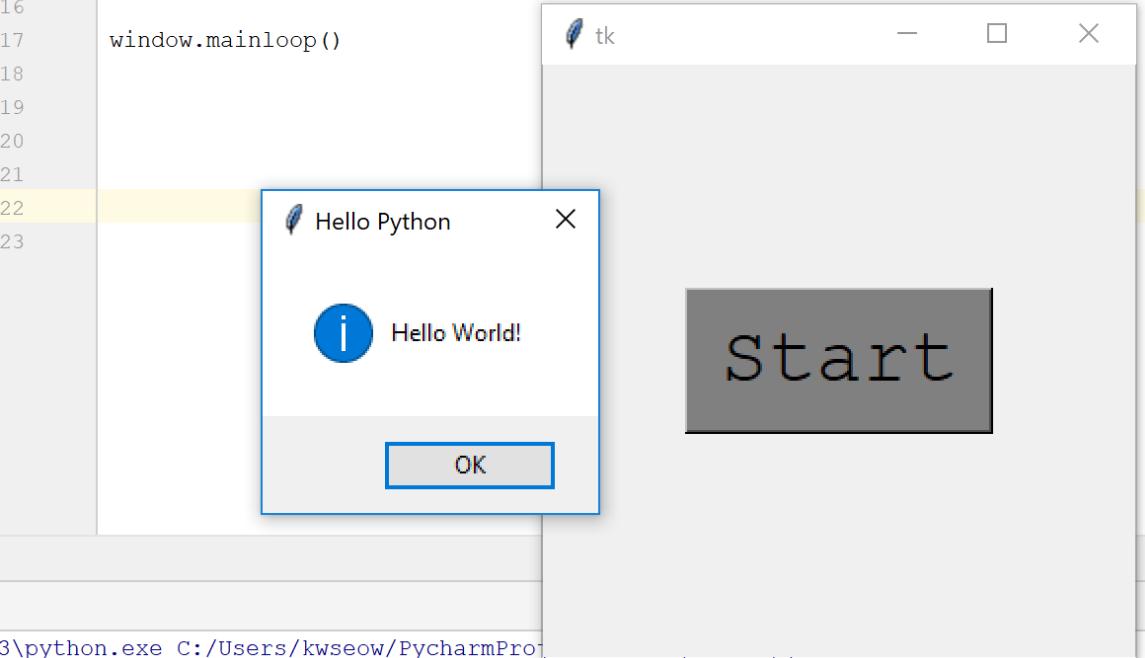
Graphical User Interface

- Getting the button to respond to click.



Callback function

```
1 import tkinter
2 import tkinter.messagebox
3
4 window = tkinter.Tk()
5
6 #set the window's size
7 window.geometry("300x300")
8
9 def displayMsg():
10     tkinter.messagebox.showinfo("Hello Python", "Hello World!")
11
12 #Add a button
13 button = tkinter.Button(window, text="Start", bg="gray", command=displayMsg)
14 button.config(font=("Courier", 30))
15 button.pack(side="top", expand=tkinter.YES)
16
17 window.mainloop()
```



3\python.exe C:/Users/kwseow/PycharmProj

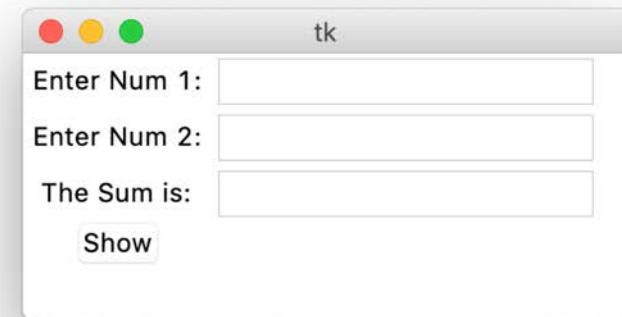
Graphical User Interface

- Use the Entry widget

```

1 import tkinter
2 import tkinter.messagebox
3
4 window = tkinter.Tk()
5
6 #set the window's size
7 window.geometry("300x300")
8
9 def show_answer():
10     Ans = int(num1.get()) + int(num2.get())
11     print(Ans)
12     ans.insert(0,Ans)
13
14 label1 = tkinter.Label(window, text = "Enter Num 1:").grid(row=0)
15 label2 = tkinter.Label(window, text = "Enter Num 2:").grid(row=1)
16 label3 = tkinter.Label(window, text = "The Sum is:").grid(row=2)
17
18 num1 = tkinter.Entry(window)
19 num2 = tkinter.Entry(window)
20 ans = tkinter.Entry(window)
21
22 num1.grid(row=0, column=1)
23 num2.grid(row=1, column=1)
24 ans.grid(row=2, column=1)
25
26 #Add a button
27 button1 = tkinter.Button(window, text="Show", bg="gray", command=show_answer)
28 button1.grid(row=4, column=0)
29
30 window.mainloop()
31
32

```



Graphical User Interface

- `filedialog` is a module with open and save dialog functions.

The screenshot shows a code editor window with a Python script titled "displayFileDialog". The script imports `tkinter`, `tkinter.messagebox`, and `tkinter.filedialog`. It creates a window and sets its size to 300x300. A function `displayFileDialog` is defined to show an open file dialog. The dialog's parameters are highlighted with a red box: `initialdir = "/", title = "Select file", filetypes = (("jpeg files", "*.jpg"), ("all files", "*.*"))`. The code also adds a button to the window. Below the code editor is a terminal window showing the output of the script's execution. The terminal output includes the Tk shell information, the Python version (3.7.3), and the command run (conda env list). At the bottom, a file dialog is displayed with the title "File...". The dialog lists "Macintosh HD" as the location and shows a filtered list of files including Applications, data, Developer, Library, System, and Users. A filter bar at the bottom of the dialog is set to "jpeg files (.jpg)".

```
1 import tkinter
2 import tkinter.messagebox
3 import tkinter.filedialog
4
5 window = tkinter.Tk()
6
7 #set the window's size
8 window.geometry("300x300")
9
10 def displayFileDialog():
11     filename = tkinter.filedialog.askopenfilename(
12         initialdir = "/",
13         title = "Select file",
14         filetypes = ( ("jpeg files", "*.jpg"),
15                       ("all files", "*.*")))
16     print (filename)
17     return
18
19 #Add a button
20 button = tkinter.Button(window, text="File...", bg="gray", command=displayFileDialog)
21 button.config(font=("Courier", 30))
22 button.pack(side="top", expand=tkinter.YES)
```

tk

Shell

and. Please wait for result

conda, Inc. | (default, Aug 26 2018, 16:30:03)
1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
pe "help", "copyright", "credits" or "license" for more information.
gui_06.py]

File...

Select file

Macintosh HD

Favorites

- kwseow
- Dropbox
- OneDrive - Republic Polytechnic
- Downloads
- DAT257x-Reinforcement Lea...
- DEV290x - Computer Vision...
- C2889C - IoT Cons Elect, He...
- C249

Name Date...modified Size

- Applications Yesterday
- data 4/7/18
- Developer 2/7/18
- Library 1/10/18
- System 21/9/18
- Users 1/10/18

Filter: jpeg files (.jpg)

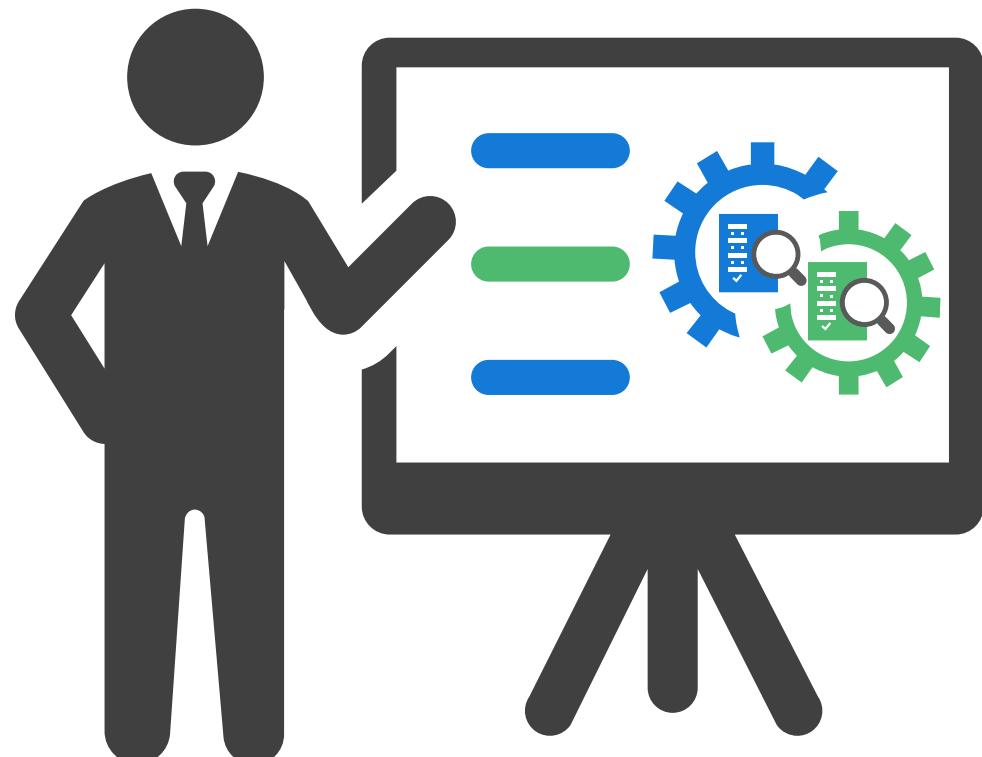
Cancel Open

Summary



Think Python is an introduction to Python programming for beginners. It starts with basic concepts of programming, and is carefully designed to define all terms when they are first used and to develop each new concept in a logical progression. Larger pieces, like recursion and object-oriented programming are divided into a sequence of smaller steps and introduced over the course of several chapters.

Think Python is a Free Book. It is available under the [Creative Commons Attribution-NonCommercial 3.0 Unported License](http://greenteapress.com/thinkpython/thinkpython.pdf), which means that you are free to copy, distribute, and modify it, as long as you attribute the work and don't use it for commercial purposes.
<http://greenteapress.com/thinkpython/thinkpython.pdf>



Day 1 Summary

- ✓ *Basics on Python*
- ✓ *Development Environment*
- ✓ *Datatypes (basic, list, dictionary)*
- ✓ *Printing*

- ✓ *Functions*
- ✓ *Time library*
- ✓ *Basic Arithmetic*
- ✓ *Getting user inputs*

- ✓ *If-Else statement*
- ✓ *For Loops*
- ✓ *Exception handling*
- ✓ *File management*
- ✓ *Graphical User Interface*

Email
seow_khee_wei@rp.edu.sg

Telegram
[@kwseow](https://t.me/kwseow)

Source code: <http://bit.ly/2GVM07s>

A stack of eight wooden blocks, each featuring a large black letter, spelling out the word "homework". The blocks are arranged in two rows: "home" on top and "work" on bottom. The colors of the blocks are blue, green, light blue, purple, red, white, pink, and light blue. The background is a vibrant yellow and orange gradient.

h o m e

w o r k

Thank you