

Dancing with Python 101

Yoochan Myung

**School of Chemistry and Molecular Biosciences
Postdoctoral Research Fellow**

Acknowledgement of Country

The University of Queensland (UQ) acknowledges the Traditional Owners and their custodianship of the lands on which we meet.

We pay our respects to their Ancestors and their descendants, who continue cultural and spiritual connections to Country.

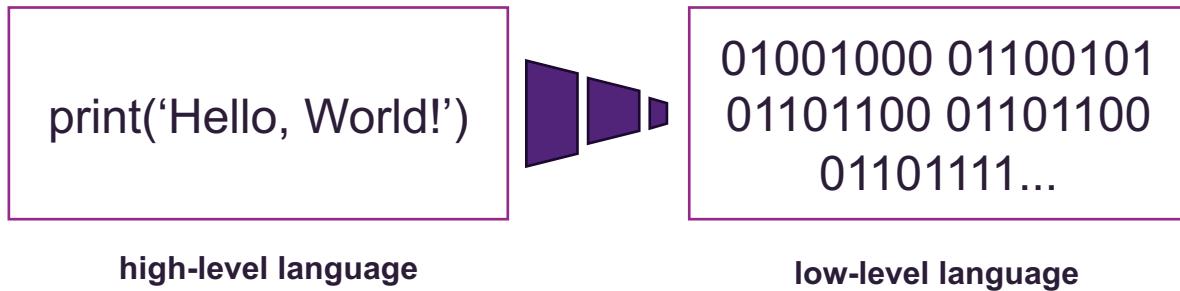
We recognise their valuable contributions to Australian and global society.



Image: Digital reproduction of *A guidance through time* by Casey Coolwell and Kyra Mancktelow

What's Python?

- A high-level programming language, 1991 (v 0.9.0) ~ current (v 3.11.4)
- Free and Open-Source



[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
<https://programmerbay.com/difference-between-high-level-language-and-low-level-language/>

Guido van Rossum

What's Python?

Why Python over other languages?

- **Easy-of-use:** great readability and straightforward syntax
- **Large and active community:** a vast community such as scientific computing, machine learning, and web development etc.
- Easily get helps via online communities such as Stack overflow, Reddit, StackExchange.



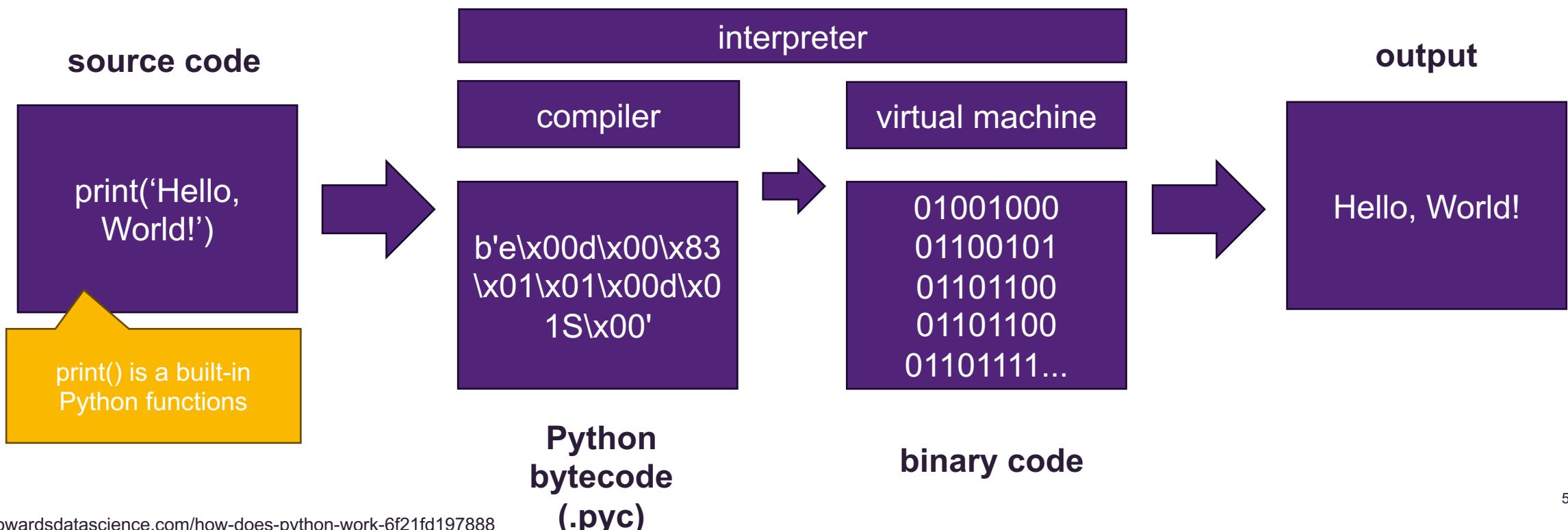
- **Extensive third-party packages:** a rich ecosystem of libraries (further installations are required)
 - The **Python Package Index (PyPI)**, the official third-party software repository for Python, provides more than 450,000 Python packages.
 - **Pandas** for data manipulation and analysis, **Scikit-learn** for machine learning, **NumPy** and **Scipy** for mathematical and scientific computing, and **Pytorch** for deep learning



What's Python?

How the Python works?

- **source code:** your code
- **bytecode:** intermediate representation of your code
- **binary code:** two-symbol machine code using “0” and “1”



What's Python?

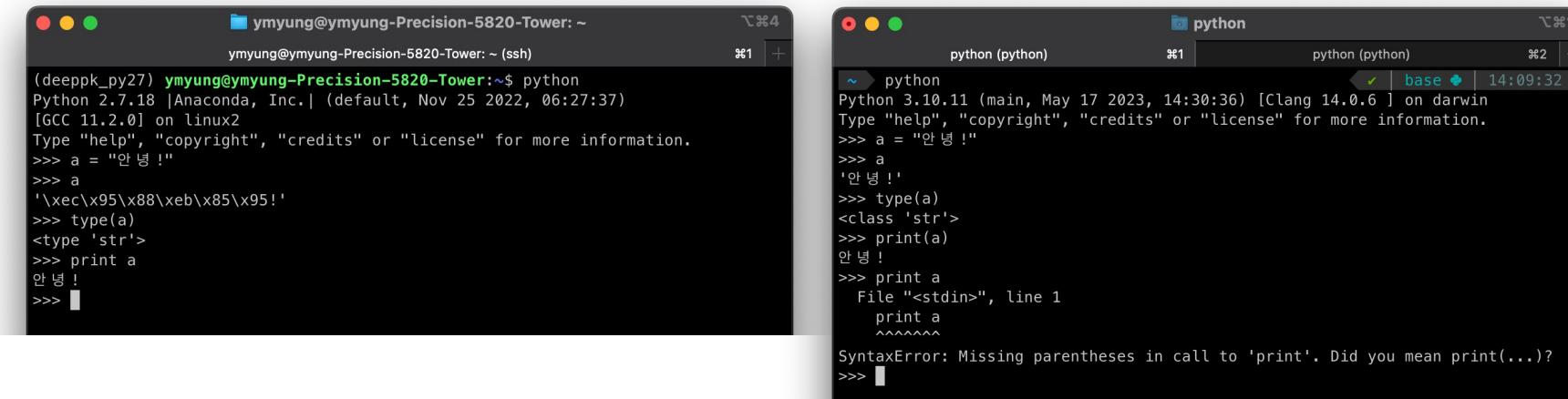
Python 2 vs Python 3

Dropped support for Python 2 in 2020.

For storage of strings, Python 2 uses ASCII and Python 3 uses UNICODE.

In Python2, “Print” is a statement not a function.

Converting Python 2 codes to Python 3 may need a lot of effort due to the incompatibility above.



The image shows two terminal windows side-by-side. The left window is titled 'y myung@y myung-Precision-5820-Tower: ~' and shows Python 2.7.18. It demonstrates the use of the 'print' statement:

```
(deppk_py27) y myung@y myung-Precision-5820-Tower:~$ python
Python 2.7.18 |Anaconda, Inc.| (default, Nov 25 2022, 06:27:37)
[GCC 11.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "안녕!"
>>> a
'안녕!'
>>> type(a)
<type 'str'>
>>> print a
안녕!
>>> 
```

The right window is titled 'python (python)' and shows Python 3.10.11. It demonstrates the use of the 'print' function:

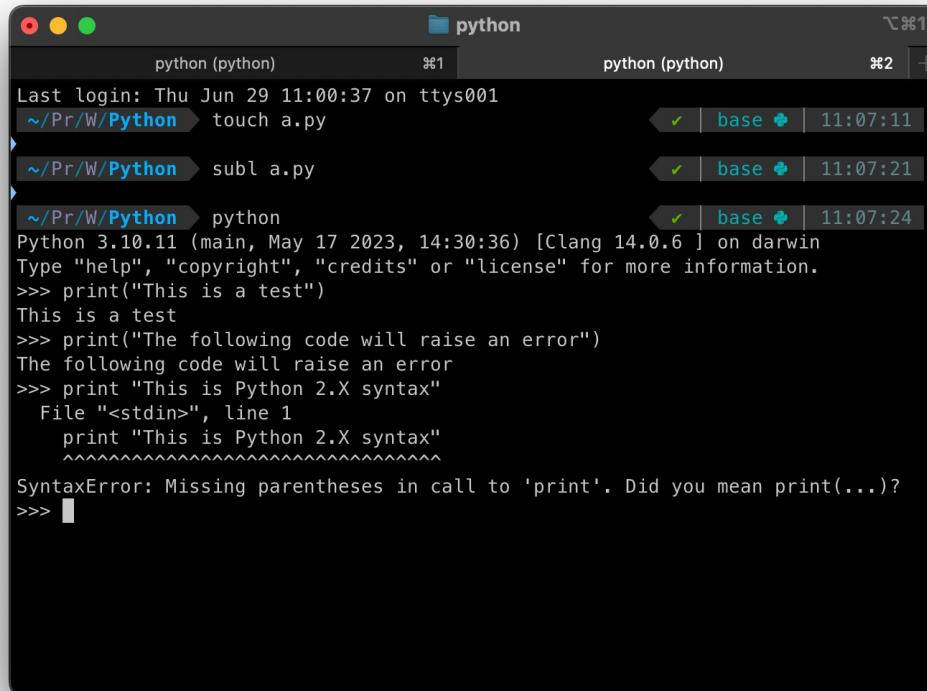
```
python (python) ~ 14:09:32
>>> python
Python 3.10.11 (main, May 17 2023, 14:30:36) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "안녕!"
>>> a
'안녕!'
>>> type(a)
<class 'str'>
>>> print(a)
안녕!
>>> print a
File "<stdin>", line 1
    print a
      ^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
>>> 
```

안녕! = Hello!

“Consider the third-party packages you intend to use. If none are required, stick with the latest versions available.”

How to start coding?

via Python IDLE* (console)



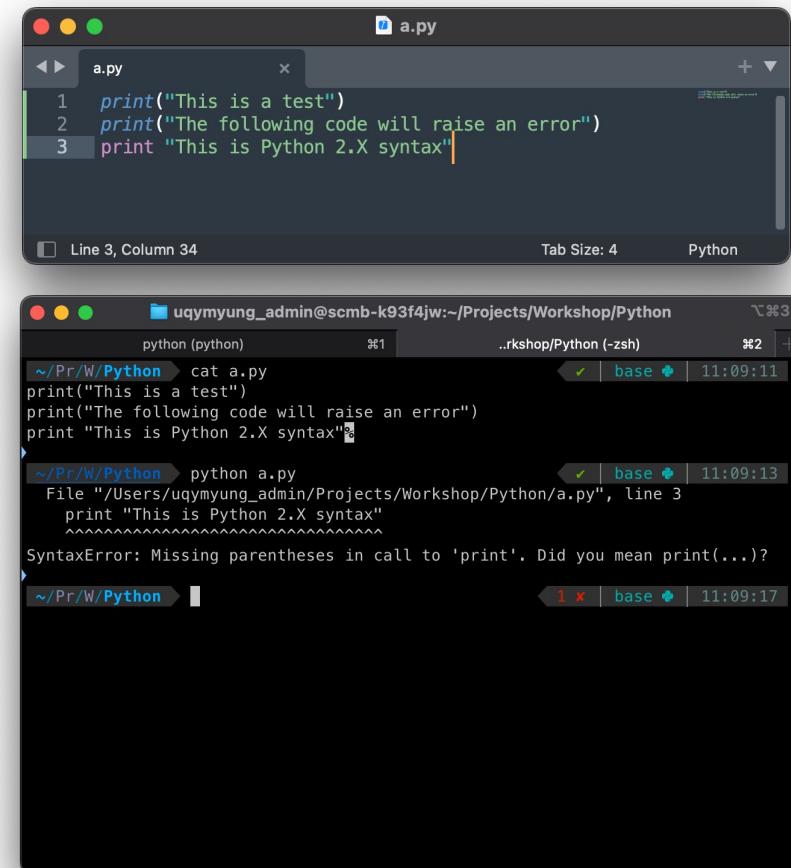
```

python (python) #1
Last login: Thu Jun 29 11:00:37 on ttys001
~/Pr/W/Python touch a.py
~/Pr/W/Python subl a.py
~/Pr/W/Python python
Python 3.10.11 (main, May 17 2023, 14:30:36) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("This is a test")
This is a test
>>> print("The following code will raise an error")
The following code will raise an error
>>> print "This is Python 2.X syntax"
  File "<stdin>", line 1
    print "This is Python 2.X syntax"
      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
>>>

```

real time,
one-by-one,
easier to experiment, test small code
segments

via Python file execution



```

a.py
1 print("This is a test")
2 print("The following code will raise an error")
3 print "This is Python 2.X syntax"

Line 3, Column 34
Tab Size: 4 Python

uqymyung_admin@scmb-k93f4jw:~/Projects/Workshop/Python
python (python) #1 .rkshop/Python (-zsh) #2 +
~/Pr/W/Python cat a.py
print("This is a test")
print("The following code will raise an error")
print "This is Python 2.X syntax"
~/Pr/W/Python python a.py
  File "/Users/uqymyung_admin/Projects/Workshop/Python/a.py", line 3
    print "This is Python 2.X syntax"
      ^^^^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
~/Pr/W/Python

```

faster,
more limited to debugging,
better to write long codes

Integrated Development Environment (IDE):

1. Syntax highlighting
2. Debugging
3. Integrated Terminal
4. ...



Visual Studio Code

```
tenfold_regressor.py
1 #!/usr/bin/env python
2 # coding: utf-8
3 # Author: Yoochan Hyung - yuchan@gmail.com
4 #
5 # -----
6 import pandas as pd
7 import numpy as np
8 import tensorflow as tf
9 import time
10 import sys
11 import os
12 import warnings
13 import argparse
14 from tensorflow import set_random_seed
15 import scipy.stats
16 import joblib
17
18 from sklearnex import patch_sklearn
19 patch_sklearn()
20
21 from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor
22 from xgboost.sklearn import XGBRegressor
23 from sklearn.gaussian_process import GaussianProcessRegressor
24 from sklearn.neighbors import KNeighborsRegressor
25 from sklearn.model_selection import KFold
26 from sklearn.metrics import mean_squared_error
27
28 from sklearn.neural_network import MLPRegressor
29
30 from sklearn.pipeline import make_pipeline
31 from sklearn.compose import make_column_transformer
32 from sklearn.preprocessing import StandardScaler, OneHotEncoder
33 from sklearn.impute import SimpleImputer
34
35 timestamp = time.strftime("%Y%m_%d%H%M")
36
37 def runM(algorithm, fname, kfold, training_pd, blind_pd, output_dir, target_label, error_type, n_cores, num_of_shuffling, random_state, no_scaling, save_model):
38     result_M = pd.DataFrame()
39     blind_result_M = pd.DataFrame()
40     blind_kfold_result = pd.DataFrame()
41     blind_result = pd.DataFrame()
42
43     train_ID = pd.DataFrame(training_pd['ID'])
44     y_input = np.array(training_pd[target_label])
45     X_input = training_pd.drop(['ID', target_label], axis=1)
46     headers = list(X_input.columns.values)
47
48     if sort_features:
49         X_input.sort_index(inplace=True)
50         X_input = X_input[headers]
51
52     if not no_scaling:
53         # numeric_features = X_input.select_dtypes('number').columns
54         # categorical_features = X_input.select_dtypes('object').columns
55
56         # numeric_transformer = make_pipeline(
57         #     SimpleImputer(strategy='median'),
58         #     StandardScaler())
59
60         # categorical_transformer = make_pipeline(
61         #     SimpleImputer(strategy='constant', fill_value='missing'),
62         #     OneHotEncoder(dtype='int', drop='if_binary')
63         # )
64
65         # col_transformer = make_column_transformer(
66         #     (numeric_transformer, numeric_features),
67         #     (categorical_transformer, categorical_features),
68         #     remainder='passthrough'
69         # )
70
71         # pipe = make_pipeline(col_transformer)
72         # pipe.fit(X_input)
73         # sc = StandardScaler()
74         # out = sc.fit_transform(X_input)
75
76     if not greedy and save_model:
77         joblib.dump(sc, os.path.join(output_dir, '{}_{}_transformer.sav'.format(timestamp, fname, algorithm)), compress=True)
78
79     X_input = pd.DataFrame(out, columns=headers)
80
81 if not isinstance(blind_pd, bool):
82     # load blind data
83     obj_bblind = joblib.load(os.path.join(output_dir, 'obj_bblind.pkl'))
84     X_bblind = pd.DataFrame(obj_bblind, columns=headers)
85     blind_pd = pd.concat([X_bblind, blind_pd[['ID', target_label]]], axis=1)
86
87 if algorithm == 'GB':
88     regressor_cv = GradientBoostingRegressor(n_estimators=300, random_state=1)
89     regressor_all = GradientBoostingRegressor(n_estimators=300, random_state=1)
90
91 elif (algorithm == 'XGBBOOST'):
92     regressor_cv = XGBRegressor(objective='reg:squarederror', n_estimators=300, random_state=1, n_jobs=n_cores)
93
94     regressor_cv = XGBRegressor(objective='reg:squarederror', n_estimators=300, random_state=1)
95
96
97
98
```



Sublime text

```
tenfold_regressor.py
1 #!/usr/bin/env python
2 # coding: utf-8
3 # Author: Yoochan Hyung - yuchan@gmail.com
4 #
5 # -----
6 import pandas as pd
7 import numpy as np
8 import tensorflow as tf
9 import time
10 import sys
11 import os
12 import warnings
13 import argparse
14 from math import sqrt
15 from scipy import stats
16 import joblib
17
18 from sklearnex import patch_sklearn
19 patch_sklearn()
20
21 from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor
22 from xgboost.sklearn import XGBRegressor
23 from sklearn.gaussian_process import GaussianProcessRegressor
24 from sklearn.neighbors import KNeighborsRegressor
25 from sklearn.model_selection import KFold
26 from sklearn.metrics import mean_squared_error
27
28 from sklearn.neural_network import MLPRegressor
29
30 from sklearn.pipeline import make_pipeline
31 from sklearn.compose import make_column_transformer
32 from sklearn.preprocessing import StandardScaler, OneHotEncoder
33 from sklearn.impute import SimpleImputer
34
35 timestamp = time.strftime("%Y%m_%d%H%M")
36
37 def runM(algorithm, fname, kfold, training_pd, blind_pd, output_dir, target_label, error_type, n_cores, num_of_shuffling, random_state, no_scaling, save_model):
38     result_M = pd.DataFrame()
39     blind_result_M = pd.DataFrame()
40     blind_kfold_result = pd.DataFrame()
41     blind_result = pd.DataFrame()
42
43     train_ID = pd.DataFrame(training_pd['ID'])
44     y_input = np.array(training_pd[target_label])
45     X_input = training_pd.drop(['ID', target_label], axis=1)
46     headers = list(X_input.columns.values)
47
48     if sort_features:
49         X_input.sort_index(inplace=True)
50         X_input = X_input[headers]
51
52     if not no_scaling:
53         # numeric_features = X_input.select_dtypes('number').columns
54         # categorical_features = X_input.select_dtypes('object').columns
55
56         # numeric_transformer = make_pipeline(
57         #     SimpleImputer(strategy='median'),
58         #     StandardScaler())
59
60         # categorical_transformer = make_pipeline(
61         #     SimpleImputer(strategy='constant', fill_value='missing'),
62         #     OneHotEncoder(dtype='int', drop='if_binary')
63         # )
64
65         # col_transformer = make_column_transformer(
66         #     (numeric_transformer, numeric_features),
67         #     (categorical_transformer, categorical_features),
68         #     remainder='passthrough'
69         # )
70
71         # pipe = make_pipeline(col_transformer)
72         # pipe.fit(X_input)
73         # sc = StandardScaler()
74         # out = sc.fit_transform(X_input)
75
76     if not greedy and save_model:
77         joblib.dump(sc, os.path.join(output_dir, '{}_{}_transformer.sav'.format(timestamp, fname, algorithm)), compress=True)
78
79     X_input = pd.DataFrame(out, columns=headers)
80
81 if not isinstance(blind_pd, bool):
82     # load blind data
83     obj_bblind = joblib.load(os.path.join(output_dir, 'obj_bblind.pkl'))
84     X_bblind = pd.DataFrame(obj_bblind, columns=headers)
85     blind_pd = pd.concat([X_bblind, blind_pd[['ID', target_label]]], axis=1)
86
87 if algorithm == 'GB':
88     regressor_cv = GradientBoostingRegressor(n_estimators=300, random_state=1)
89     regressor_all = GradientBoostingRegressor(n_estimators=300, random_state=1)
90
91 elif (algorithm == 'XGBBOOST'):
92     regressor_cv = XGBRegressor(objective='reg:squarederror', n_estimators=300, random_state=1)
93
94     regressor_cv = XGBRegressor(objective='reg:squarederror', n_estimators=300, random_state=1)
95
96
97
98
```



Vim

```
tenfold_regressor.py
1 #!/usr/bin/env python
2 # coding: utf-8
3 # Author: Yoochan Hyung - yuchan@gmail.com
4 #
5 # -----
6 import pandas as pd
7 import numpy as np
8 import tensorflow as tf
9 import time
10 import sys
11 import os
12 import warnings
13 import argparse
14 from math import sqrt
15 from scipy import stats
16 import joblib
17
18 from sklearnex import patch_sklearn
19 patch_sklearn()
20
21 from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor
22 from xgboost.sklearn import XGBRegressor
23 from sklearn.gaussian_process import GaussianProcessRegressor
24 from sklearn.neighbors import KNeighborsRegressor
25 from sklearn.model_selection import KFold
26 from sklearn.metrics import mean_squared_error
27
28 from sklearn.neural_network import MLPRegressor
29
30 from sklearn.pipeline import make_pipeline
31 from sklearn.compose import make_column_transformer
32 from sklearn.preprocessing import StandardScaler, OneHotEncoder
33 from sklearn.impute import SimpleImputer
34
35 timestamp = time.strftime("%Y%m_%d%H%M")
36
37 def runM(algorithm, fname, kfold, training_pd, blind_pd, output_dir, target_label, error_type, n_cores, num_of_shuffling, random_state, no_scaling, save_model, sort_features, greedy=False):
38     result_M = pd.DataFrame()
39     blind_result_M = pd.DataFrame()
40     blind_kfold_result = pd.DataFrame()
41     blind_result = pd.DataFrame()
42
43     train_ID = pd.DataFrame(training_pd['ID'])
44     y_input = np.array(training_pd[target_label])
45     X_input = training_pd.drop(['ID', target_label], axis=1)
46     headers = list(X_input.columns.values)
47
48     if sort_features:
49         X_input.sort_index(inplace=True)
50         X_input = X_input[headers]
51
52     if not no_scaling:
53         # numeric_features = X_input.select_dtypes('number').columns
54         # categorical_features = X_input.select_dtypes('object').columns
55
56         # numeric_transformer = make_pipeline(
57         #     SimpleImputer(strategy='median'),
58         #     StandardScaler())
59
60         # categorical_transformer = make_pipeline(
61         #     SimpleImputer(strategy='constant', fill_value='missing'),
62         #     OneHotEncoder(dtype='int', drop='if_binary')
63         # )
64
65         # col_transformer = make_column_transformer(
66         #     (numeric_transformer, numeric_features),
67         #     (categorical_transformer, categorical_features),
68         #     remainder='passthrough'
69         # )
70
71         # pipe = make_pipeline(col_transformer)
72         # pipe.fit(X_input)
73         # sc = StandardScaler()
74         # out = sc.fit_transform(X_input)
75
76     if not greedy and save_model:
77         joblib.dump(sc, os.path.join(output_dir, '{}_{}_transformer.sav'.format(timestamp, fname, algorithm)), compress=True)
78
79     X_input = pd.DataFrame(out, columns=headers)
80
81 if not isinstance(blind_pd, bool):
82     # load blind data
83     obj_bblind = joblib.load(os.path.join(output_dir, 'obj_bblind.pkl'))
84     X_bblind = pd.DataFrame(obj_bblind, columns=headers)
85     blind_pd = pd.concat([X_bblind, blind_pd[['ID', target_label]]], axis=1)
86
87 if algorithm == 'GB':
88     regressor_cv = GradientBoostingRegressor(n_estimators=300, random_state=1)
89     regressor_all = GradientBoostingRegressor(n_estimators=300, random_state=1)
90
91 elif (algorithm == 'XGBBOOST'):
92     regressor_cv = XGBRegressor(objective='reg:squarederror', n_estimators=300, random_state=1)
93
94     regressor_cv = XGBRegressor(objective='reg:squarederror', n_estimators=300, random_state=1)
95
96
97
98
```





Jupyter Notebook

- A unique combination of interactive code execution, documentation and data analysis features.
- Run on Google CoLab server
- Easy to share and less compatibility issues

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell at the top contains three print statements. The first two are executed successfully, while the third causes a SyntaxError due to Python 2 syntax. The error message is displayed below the cell. The code cell below it runs without errors, printing the same three lines. The bottom cell also runs without errors, demonstrating the use of Python 2 syntax.

```
0s
print("This is a test")
print("The following code will raise an error")
print "This is Python 2.X syntax"

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?


[4]: print("This is a test")
      print("The following code will raise an error")

This is a test
The following code will raise an error

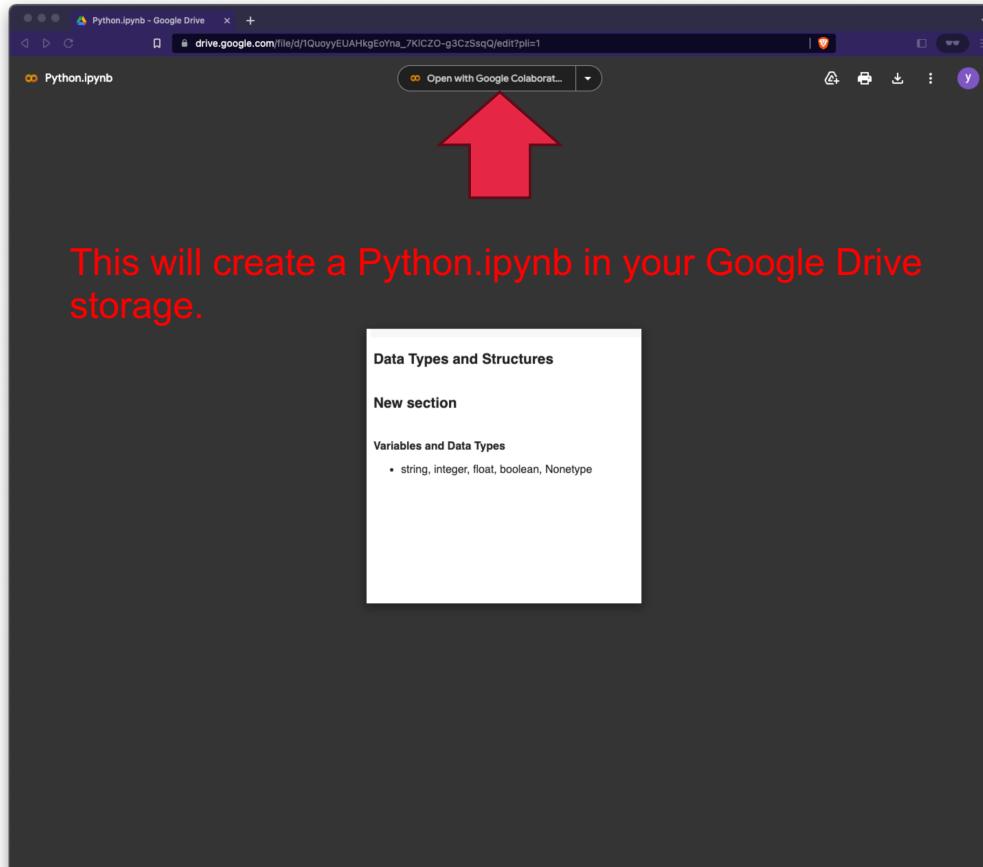
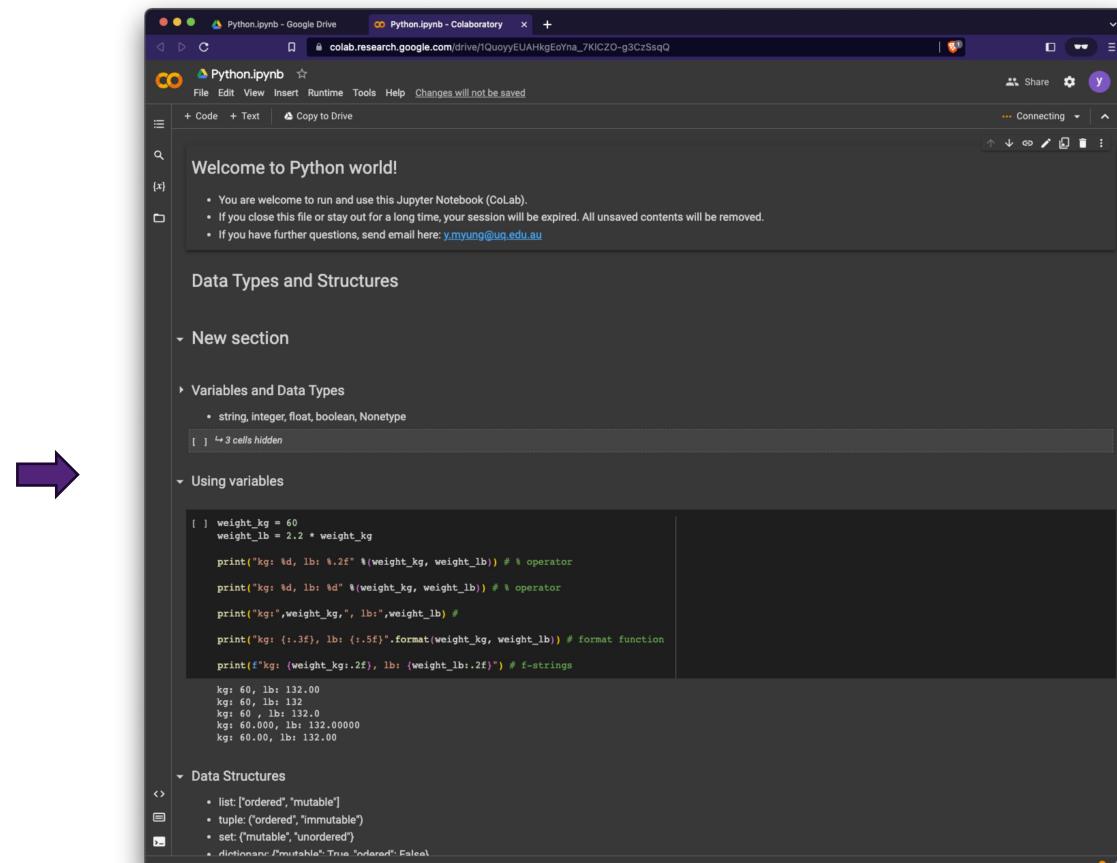
0s
print "This is Python 2.X syntax"

File "<ipython-input-5-dab6d74b1348>", line 1
      print "This is Python 2.X syntax"
^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

The screenshot shows the Google Colab homepage. It features a section for 'Curated Notebooks' with a brief description. Below this, there's a section for 'AI & Machine Learning' with several thumbnail previews of different notebooks. Each thumbnail includes a small image, the notebook title, and a brief description.

- Curated Notebooks**
Here you'll find a series of instructive and educational notebooks organized by topic areas.
- AI & Machine Learning**
- Music Transcription with Transformers**
Interactive demo of a few music transcription models created by Google's Magenta team. You can upload audio and have one of our models automatically transcribe it.
- Generating Music with Transformers**
This Colab notebook lets you play with pretrained Transformer models for piano music generation, based on the Music Transformer model introduced by Huang et al. in 2018.
- Retraining an Image Classifier**
This Colab demonstrates how to build a Keras model for classifying five species of flowers by using a pre-trained TensorFlow SavedModel from TensorFlow Hub for image feature extraction, trained on the much larger and more general ImageNet...
- Text Classification with Movie Reviews**
This notebook classifies movie reviews as positive or negative using the text of the review. This is an example of binary—or two-class—classification, an important and widely applicable kind of machine learning problem.
- Multilingual Universal Sentence Encoder Q&A Retrieval**
Demo using Universal Encoder Multilingual Q&A model for question-answer retrieval of text, illustrating the use of question_encoder and response_encoder of the model.
- Create and Train a Custom RL Agent**
This colab demonstrates how to create a variant of a provided agent (Example 1) and how to create a new agent from scratch (Example 2).
- Visualize RL Agent Training on TensorBoard**
This colab allows you to easily view the trained baselines with Tensorboard (even if you don't have Tensorboard on your local machine). Simply specify the game you would like to visualize and then run the cells in order.
- Hyperparameter Tuning with TensorBoard**
The HPparams dashboard in TensorBoard provides several tools:

How to use Google CoLab

```

Welcome to Python world!
• You are welcome to run and use this Jupyter Notebook (CoLab).
• If you close this file or stay out for a long time, your session will be expired. All unsaved contents will be removed.
• If you have further questions, send email here: y.myung@uq.edu.au

Data Types and Structures

New section

Variables and Data Types
• string, integer, float, boolean, Nonetype
[ ] ↗ 3 cells hidden

Using variables

[ ] weight_kg = 60
weight_lb = 2.2 * weight_kg
print("kg: %d, lb: %.2f" %(weight_kg, weight_lb)) # % operator
print("kg: %d, lb: %d" %(weight_kg, weight_lb)) # % operator
print("kg:",weight_kg," lb:",weight_lb) #
print("kg: {:.3f}, lb: {:.5f}".format(weight_kg, weight_lb)) # format function
print(f"kg: {weight_kg:.2f}, lb: {weight_lb:.2f}") # f-strings

kg: 60, lb: 132.00000
kg: 60, lb: 132.0
kg: 60.0, lb: 132.0
kg: 60.000, lb: 132.00000
kg: 60.00, lb: 132.00000

Data Structures
• list: ['ordered', 'mutable']
• tuple: ('ordered', 'immutable')
• set: {'mutable', 'unordered'}
• dictionary: 'mutable', 'ordered', 'mutable'

```

Don't worry, it's a copy of the original Python.ipynb and saving from your files will NOT affect to the original file.

TIME FOR A BREAK

Please be ready for using Google CoLab and ChatGPT!

<https://shorturl.at/hKOZ5>

<https://openai.com/blog/chatgpt>

Python Data Types and Structures

“Hello”

1.61

None

1523

True

[1, 2, 1, “hi”, 4.44, “5”, None, True, None]

(1, 2, 1, “hi”, 4.44, “5”, None, True, None)

{"first name": "Yoochan", "last name": "Myung"}

{1, 2, 4.44, “hi”, None, “5”}

Python Variables and Data Types

Variables: we need to assign (=) our data (“Hello, World!”) to a variable (v1).

- can contain letters, digits and underscores(_).
- Case-sensitive (v1 ≠ V1)
- First character must be letter or an underscore(_).
- Cannot use Python reserved keywords as variable names

reserved keywords

False, class, finally, is, return
 None, continue, for, lambda, try
 True, def, from, nonlocal, while
 and, del, global, not, with
 as, elif, if, or, yield
 assert, else, import, pass
 break, except, in, raise

built-in functions

list, set, dict, tuple, str, int, float,
 bool

Types of data

- integer number: 1, 10, 100
- floating point number: 3.14
- string: “Hi”, “1”, “3.14”
- boolean: True (1), False (0)
- Nontype: None

“Hello”	1.61
None	
1523	True

Python Variables and Data Types

- Q1. Assign the five data types into different variables.
Q2. Print the variables and their types using `print()` and `type()`.

“Hello,
World!”

1.23

None

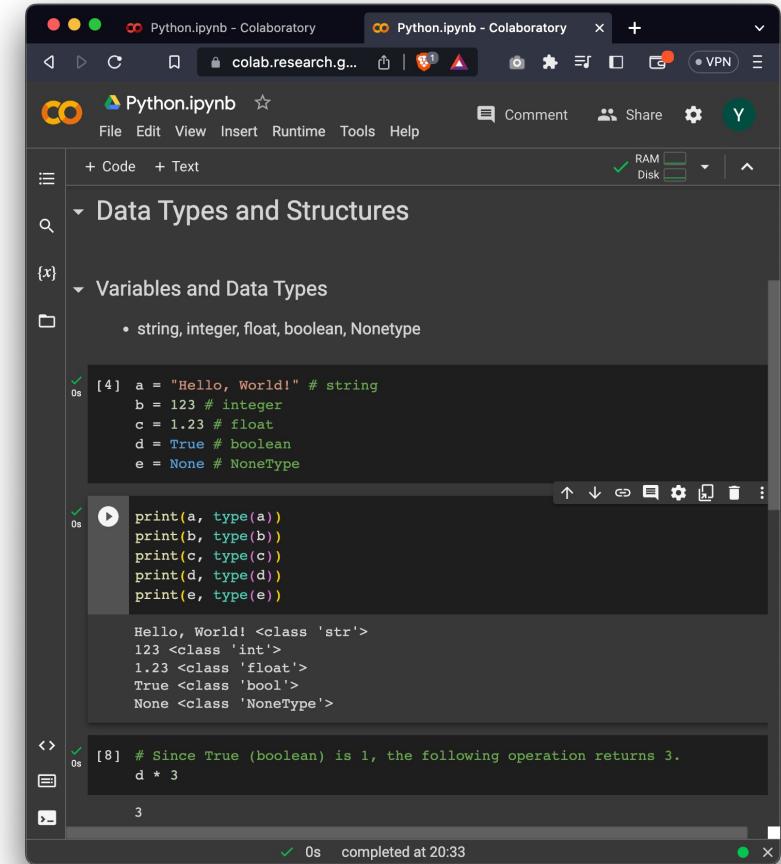
123

True

Python Variables and Data Types

- Q1. Assign the five data types into different variables.
Q2. Print the variables and their types using `print()` and `type()`.

```
"Hello,  
World!"           1.23  
  
None  
  
123              True
```



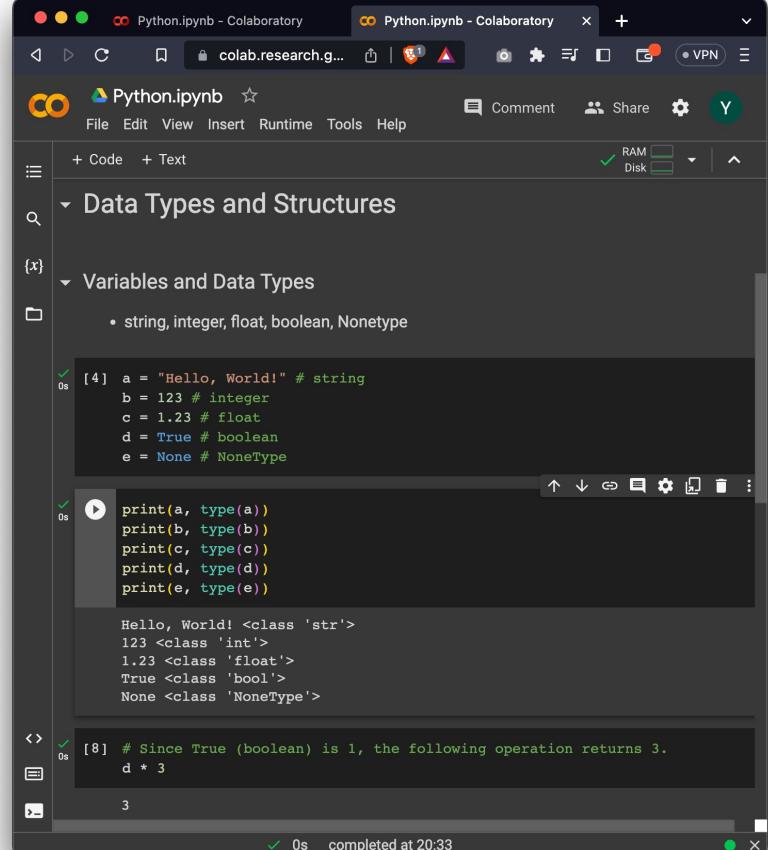
The screenshot shows a Google Colab notebook titled "Python.ipynb - Colaboratory". The sidebar is collapsed. The code cell at index 4 defines variables `a` through `e` with their respective values and types. The subsequent cell at index 5 prints each variable and its type using `print(type(a))` for each variable. The output shows the type of each variable: `a` is a string, `b` is an integer, `c` is a float, `d` is a boolean, and `e` is NoneType. A final cell at index 8 demonstrates that multiplying the boolean value `True` by 3 results in 3.

```
[4] a = "Hello, World!" # string  
      b = 123 # integer  
      c = 1.23 # float  
      d = True # boolean  
      e = None # NoneType  
  
[5] print(a, type(a))  
    print(b, type(b))  
    print(c, type(c))  
    print(d, type(d))  
    print(e, type(e))  
  
Hello, World! <class 'str'>  
123 <class 'int'>  
1.23 <class 'float'>  
True <class 'bool'>  
None <class 'NoneType'>  
  
[8] # Since True (boolean) is 1, the following operation returns 3.  
      d * 3  
  
3
```

Python Variables and Data Types

- Q1. Assign the five data types into different variables.
 Q2. Print the variables and their types using `print()` and `type()`.

```
a = str("Hello, World!")
b = int(123)
b = int("123")
c = float(1.23)
c = float("1.23")
# for False
d = bool(0)
d = bool(False)
d = bool(None)
d = bool(100-100)
d = bool(0/100)
# for True
d = bool("ANYTHING except 0,False, None")
```



The screenshot shows a Google Colab notebook titled "Python.ipynb - Colaboratory". The sidebar has sections for "Data Types and Structures" and "Variables and Data Types". In the code editor, cell [4] contains variable assignments:

```
[4] a = "Hello, World!" # string
    b = 123 # integer
    c = 1.23 # float
    d = True # boolean
    e = None # NoneType
```

Cell [5] contains the print statements:

```
[5] print(a, type(a))
      print(b, type(b))
      print(c, type(c))
      print(d, type(d))
      print(e, type(e))
```

The output of cell [5] is:

```
Hello, World! <class 'str'>
123 <class 'int'>
1.23 <class 'float'>
True <class 'bool'>
None <class 'NoneType'>
```

Cell [8] contains a multiplication operation:

```
[8] # Since True (boolean) is 1, the following operation returns 3.
      d * 3
```

The output of cell [8] is:

```
3
```

At the bottom, it says "0s completed at 20:33".

Python Variables and Data Types

Using operators with data types

Try operators (+, -, /, *) with the five variables.

Q3. What combinations of operators and operands are valid?



```
weight_kg = 60
1 weight_kg = 60
2
3 weight_lb = 2.2 * weight_kg
4
```

Line 4, Column 1 Tab Size: 4 Python



```
untitled
1
2 guest_id = '002'
3 guest_id = 'Guest' + gueest_id
4
5
```

Line 5, Column 1 Tab Size: 4 Python

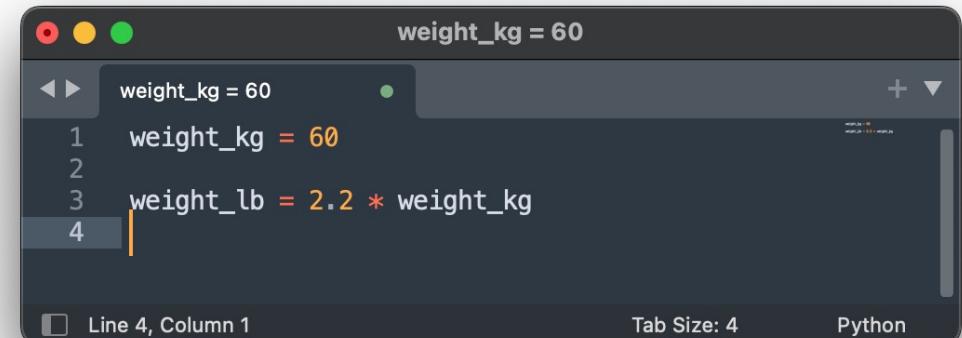
Python Variables and Data Types

Using operators with data types

Try operators (+, -, /, *) with the five variables.

Q3. What combinations of operators and operands are valid?

- ‘int or float’ and ‘int or float’ : +, -, /, *
- ‘str’ and ‘str’: +
- ‘str’ and ‘int’: *
- `int or float` and `bool` : +, -, *, / (except division by False (0))
 - because True == 1, False == 0



```
weight_kg = 60
1 weight_kg = 60
2
3 weight_lb = 2.2 * weight_kg
4
```

Line 4, Column 1 Tab Size: 4 Python



```
untitled
1 guest_id = '002'
2 guest_id = 'Guest' + gueest_id
3
4
5
```

Line 5, Column 1 Tab Size: 4 Python

Python Variables and Data Types

Numeric Types – int, float and complex

Operation	Meaning	Result
<code>1 + 4</code>	sum	5
<code>1 - 4</code>	difference	-3
<code>1 * 4</code>	product	4
<code>1 / 4, 11 / 5</code>	quotient of 1 and 4	0.25, 2.2
<code>1 // 4, 11 // 5</code>	floored quotient of 1 and 4	0, 2, 2
<code>1 % 4</code>	remainder of 1 / 4	1
<code>abs(-4)</code>	absolute	4
<code>int(3.14), int(3.9)</code>	convert to integer	3
<code>float(3)</code>	convert to floating point	3.0
<code>pow(2,3)</code>	2 to the power 3	8
<code>2 ** 3</code>	2 to the power 3	8

Python Variables and Data Types

Python String Formatting

The basic `print()` function does not provide direct control over the decimal point precision of floating-point numbers.

% operator:

- `'%s'` for string, `'%d'` for integer and `'%f'` for float.

`format()` : since Python 3.0

f-strings: since Python 3.6

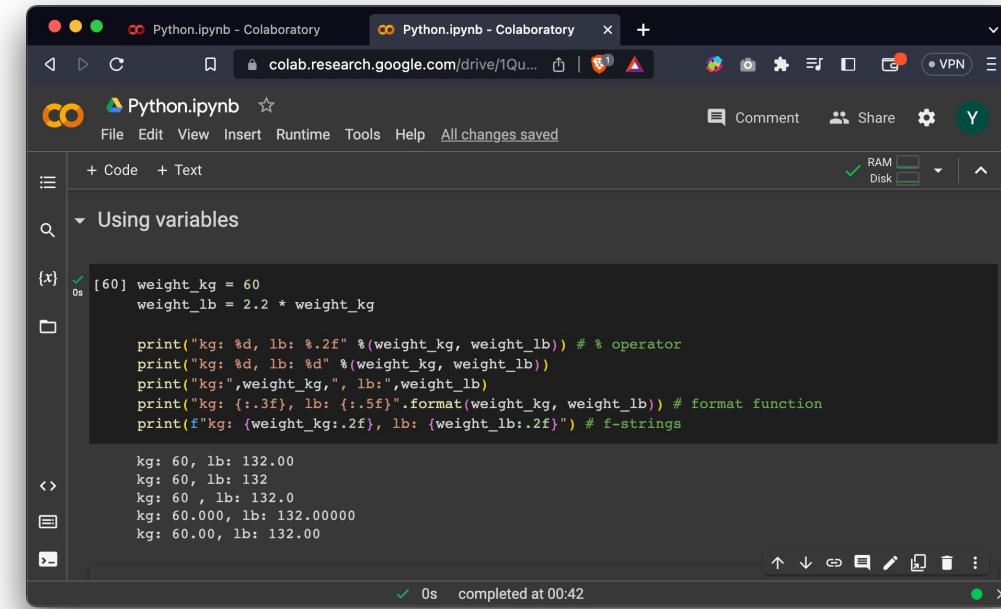
- `'{:d}'` for integer, `'{:.f}'` for a floating-point number with two decimal places

{:.2f}

`:` the start of the format specifier

`2` specifies the precision or the number of decimal places to display

`f` the type specifier for floating-point numbers



The screenshot shows a Google Colab notebook titled "Python.ipynb". The code cell contains the following Python code:

```
[60] weight_kg = 60
      weight_lb = 2.2 * weight_kg

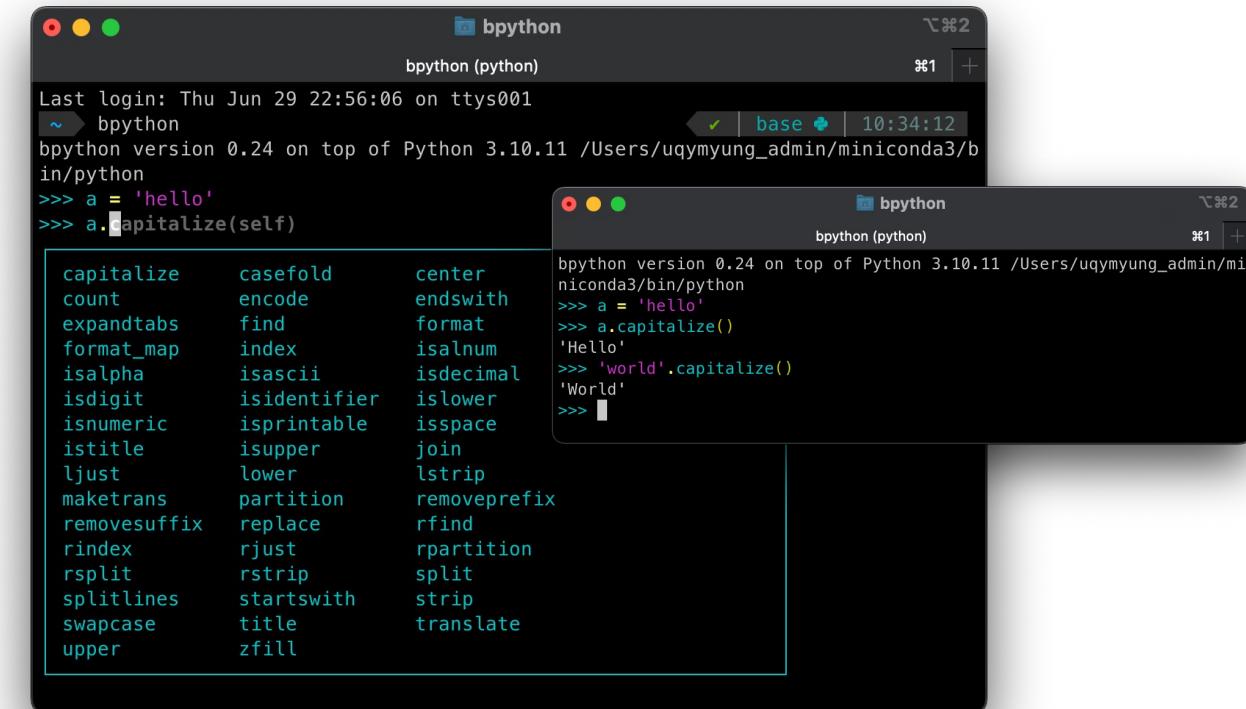
      print("kg: %d, lb: %.2f" %(weight_kg, weight_lb)) # % operator
      print("kg: %d, lb: %d" %(weight_kg, weight_lb))
      print("kg:", weight_kg, ", lb:", weight_lb)
      print("kg: {:.3f}, lb: {:.5f}".format(weight_kg, weight_lb)) # format function
      print(f"kg: {weight_kg:.2f}, lb: {weight_lb:.2f}") # f-strings
```

The output cell shows the results of the prints:

```
kg: 60, lb: 132.00
kg: 60, lb: 132
kg: 60 , lb: 132.0
kg: 60.000, lb: 132.00000
kg: 60.00, lb: 132.00
```

String functions

function	example	output
str.capitalize(self)		Hello
str.upper(self)		HELLO
str.lower(self)		hello
str.count(obj, sub[, start[, end]])	str.count('l') str.count('l', 4)	2 1
str.isalpha(self)		True
str.isnumeric(self)		False
str.strip(chars)		hello
str.lstrip(chars)	str.lstrip('h')	ello
str.rstrip(chars)	str.rstrip('h')	hello
str.startswith(obj, [, start[, end]])	str.startswith('h')	True
str.endswith(obj, [, start[, end]])	str.endswith('h')	False
str.replace(self, old, new, count)	str.replace('l','a',2)	heaaoo



The image shows two side-by-side Python consoles. The left console shows the execution of `a = 'hello'` and `a.capitalize()`, resulting in 'Hello'. The right console shows the execution of `'world'.capitalize()`, resulting in 'World'. Below the consoles is a large list of `str` method names, each preceded by a small icon.

```

Last login: Thu Jun 29 22:56:06 on ttys001
~ > bpython
bpython version 0.24 on top of Python 3.10.11 /Users/uqymyung_admin/miniconda3/bin/python
>>> a = 'hello'
>>> a.capitalize()
'Hello'

bpython version 0.24 on top of Python 3.10.11 /Users/uqymyung_admin/miniconda3/bin/python
>>> a = 'hello'
>>> a.capitalize()
'Hello'
>>> 'world'.capitalize()
'World'
>>> 

capitalise    casefold      center        endswith     format       isalnum      isdecimal   islower      lstrip      removeprefix
count         encode        encode        find         format       isalpha      isdecimal   islower      join       removeprefix
expandtabs    find          encode        index        isalnum      isascii      isidentifier islower      join       rfind
format_map   index         encode        isascii      isalpha      isdecimal   isidentifier islower      join       rpartition
isalpha       isascii       encode        isidentifier isdecimal   islower      isspace      join       partition
isdigit      isdecimal     encode        isprintable islower      islower      isspace      join       replace
isnumeric    isidentifier  encode        isprintable islower      islower      isspace      join       rstrip
istitle      islower       encode        isupper     islower      islower      isspace      join       split
ljust        lower         encode        isupper     islower      islower      isspace      join       strip
maketrans    lower         encode        isupper     islower      islower      isspace      join       translate
removeprefix replace        encode        rjust       islower      islower      isspace      join       strip
rindex      rjust         encode        rstrip      islower      islower      isspace      join       zfill
rsplit      rstrip        encode        startswith islower      islower      isspace      join       zfill
splitlines  startswith   encode        swapcase    islower      islower      isspace      join       zfill
swapcase    title         encode        title      islower      islower      isspace      join       zfill
upper       title         encode        upper     islower      islower      isspace      join       zfill
zfill
  
```

Q4. Test `str.isalpha()` on 'hello', 'hello!', 'hell0', and ' hello'.

Python Data Structures

- **List:** ordered and mutable collections of elements
 - [1, 2, 1, "hi", 4.44, "5", None, True, None]
- **Tuple:** ordered and immutable collections of elements
 - (1, 2, 1, "hi", 4.44, "5", None, True, None)
- **Set:** unordered and mutable collections of unique elements
 - {1, 2, 4.44, "hi", None, "5"}



```
list_a = [1, 2, 1, "hi", 4.44, "5", None, True, None]

tuple_a = (1, 2, 1, "hi", 4.44, "5", None, True, None)

set_a = {1, 2, 4.44, "hi", None, "5"}

# or

list_a = [1, 2, 1, "hi", 4.44, "5", None, True, None]

tuple_a = tuple(list_a)
set_a = set(list_a) # or set(tuple_a)
```

Q5. What happens if you convert a list,
["True", True, False, 1, 0]
to a set using `set()`?

List

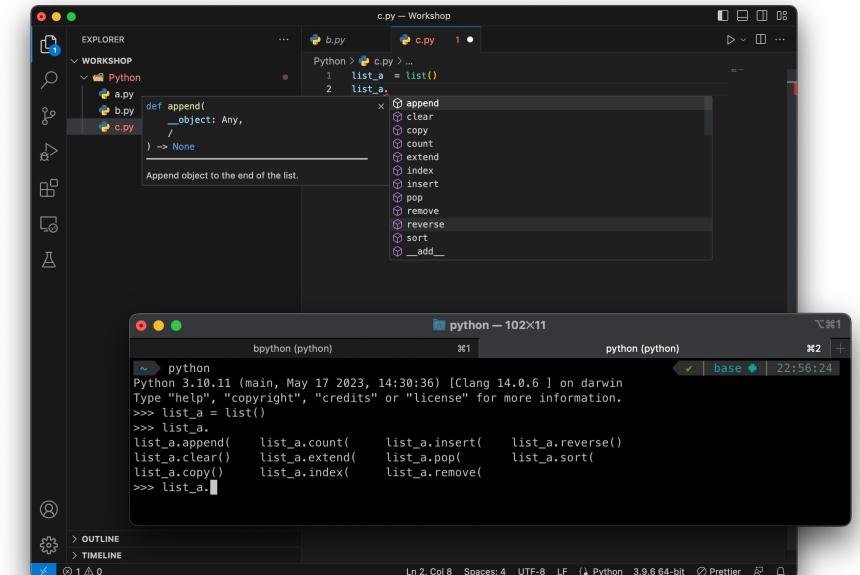
List is a mutable and ordered sequence class.

Mutable and immutable classes have some common functions.

- `append(self, object)`: Append object to the end of the list.
- `extend(self, iterable)`: Extend list by appending elements from the iterable.
- `insert(self, index, object)`: Insert object before index
- `clear(self)`: Remove all items from list.
- `index(self, value, start, stop)`: Return first index of value or ValueError if the value is not present.
- `remove(self, value)`: Remove first occurrence of value or raise ValueError if the value is not present.
- `sort(self, *, key=None, reverse=False)`: Sort the list in ascending order

```
● ○ ●

list_a = [1, 2, 1, "hi", 4.44, "5", None, True, None]
list_b = [None, True, None, 1, 2, 1, "hi", 4.44, "5"]
```



Q6. What's the difference between append and extend?

List

Q7. What happens if the built-in functions are used as variables?

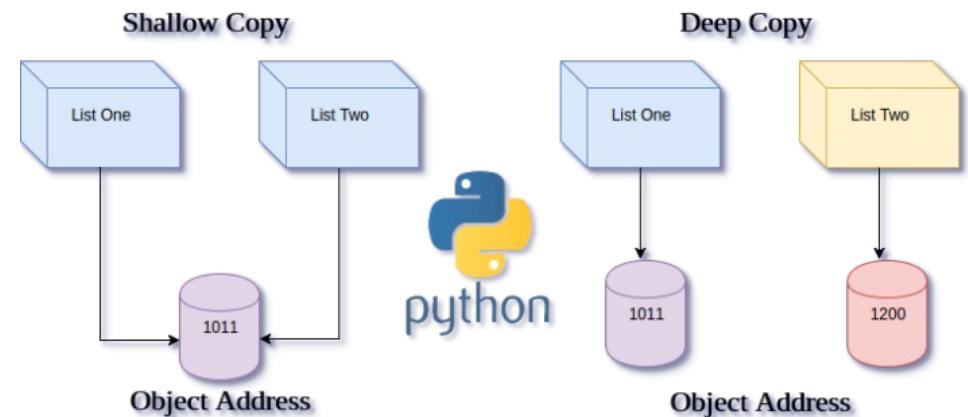
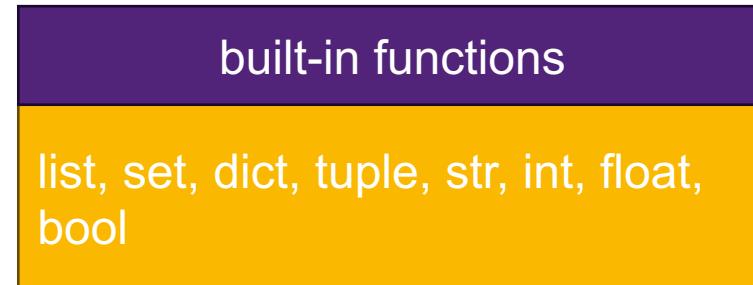
How to copy a list object?

1. Using `=` operator [assigning]
2. Using `list.copy()` [shallow copy]
3. Using `copy` module [deep copy]

A shallow copy in Python creates a new object that references the original elements of the copied object.

When you have a list of lists and create a shallow copy, it copies the references of the inner lists rather than creating new independent copies. As a result, changes made to the values within the inner lists will be reflected in both the original list and the shallow copy.

???



Range

Produces a sequence of integers from start (inclusive) to stop (exclusive) by step.

`range(stop)` or `range(start, stop[, step])`

`range(i, j)` or `range(i, j, 1)` produces $i, i+1, i+2, \dots, j-1$

- `list(range(0, 4)) = [0, 1, 2, 3]`
- `list(range(4)) = [0, 1, 2, 3]`
- `list(range(0, 4, 1)) = [0, 1, 2, 3]`

This works only with integers, doesn't support float type. (Use `numpy.arange()`)

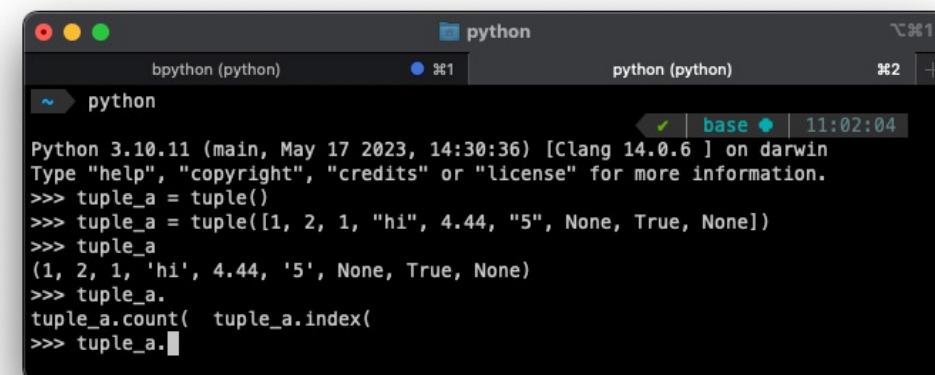
Tuple

Tuple is an immutable sequence class.

- `count(self, value)`: Return number of occurrences of value.
- `index(self, value, start, stop)`: Return first index of value.

Q8. Can we add elements to an existing tuple?

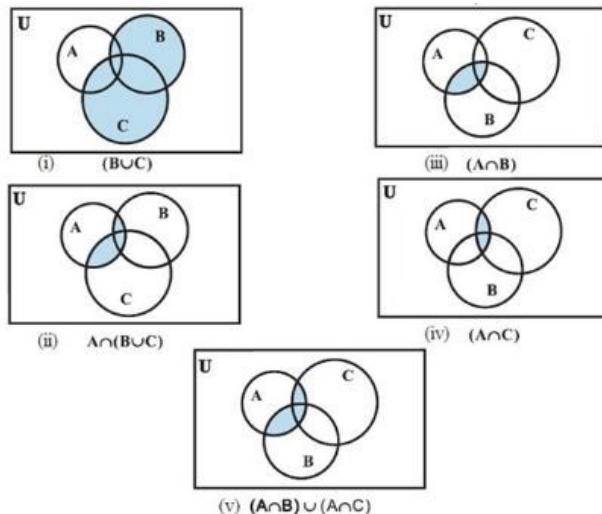
```
list_a = [1, 2, 1, "hi", 4.44, "5", None, True, None]
tuple_a = (1, 2, 1, "hi", 4.44, "5", None, True, None)
# or
tuple_a = tuple(list_a)
```



```
python
bpython (python)  python (python)
~ python
python
base 11:02:04
Python 3.10.11 (main, May 17 2023, 14:30:36) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> tuple_a = tuple()
>>> tuple_a = tuple([1, 2, 1, "hi", 4.44, "5", None, True, None])
>>> tuple_a
(1, 2, 1, 'hi', 4.44, '5', None, True, None)
>>> tuple_a.
tuple_a.count(  tuple_a.index(
>>> tuple_a.■
```

Set

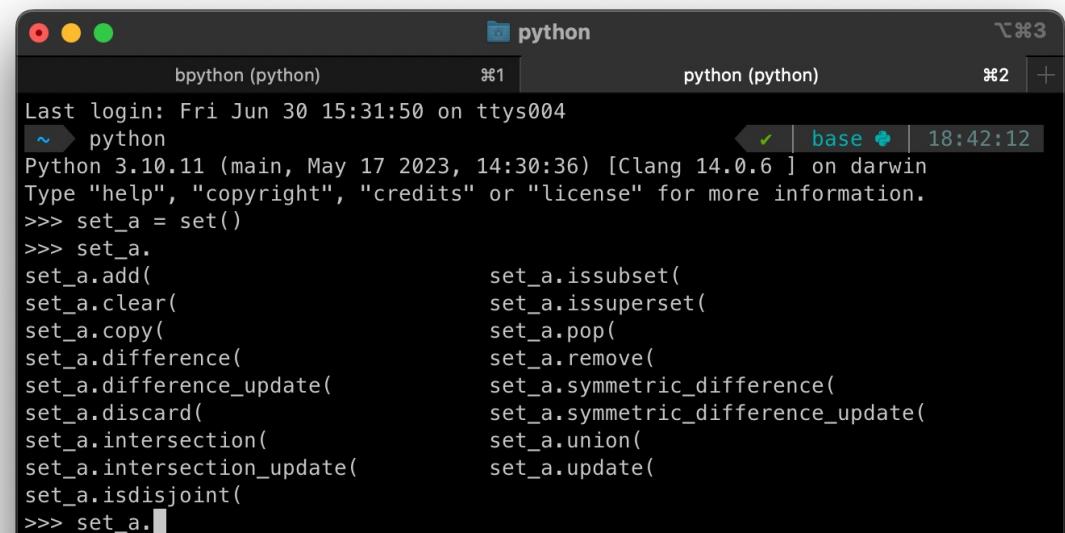
Set is an unordered collection of distinct hashable objects.



The **hash()** function generates a hash value based on the contents of an object, while the **id()** function returns a unique identifier based on the memory address of an object.

<https://guidancecorner.com/set-theory/>

Hash value: `hash(object)` returns an integer that is unique to the object.
 Same hash values → same objects.
 This can be used for data lookup, data integrity checks and object uniqueness testing.



A screenshot of a Mac OS X terminal window titled "python". It shows two tabs: "bpython (python)" and "python (python)". The "python (python)" tab contains the following Python session:

```
Last login: Fri Jun 30 15:31:50 on ttys004
python
Python 3.10.11 (main, May 17 2023, 14:30:36) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> set_a = set()
>>> set_a.
set_a.add(
set_a.clear(
set_a.copy(
set_a.difference(
set_a.difference_update(
set_a.discard(
set_a.intersection(
set_a.intersection_update(
set_a.isdisjoint(
>>> set_a.
```

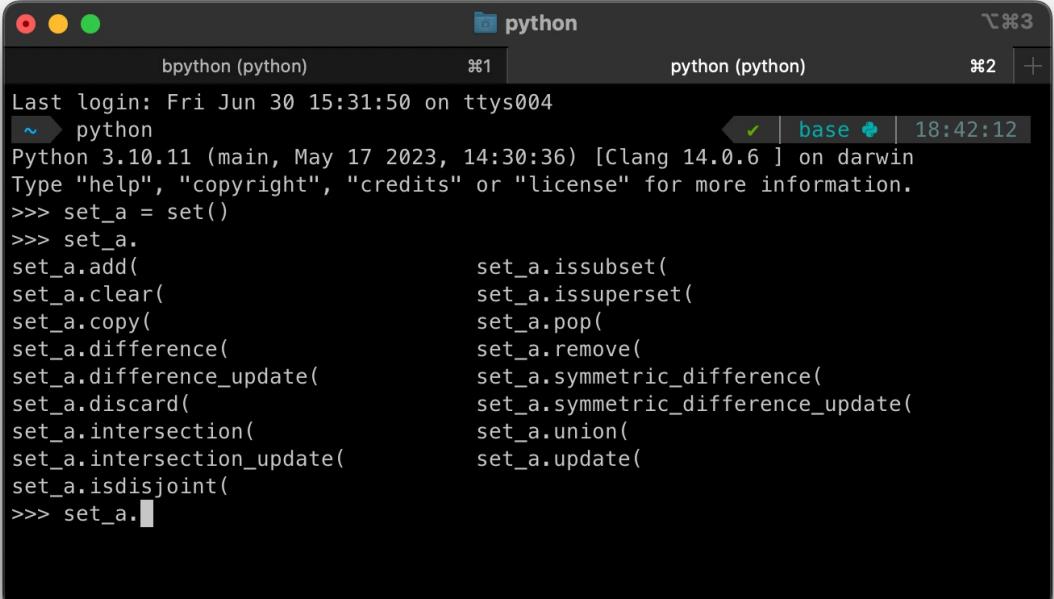
Set

Set is an unordered collection of distinct hashable objects.

Q9. What is the hash value of 'A' and 'a'?

- `set_A.union(set_B) = set_A | set_B`
- `set_A.difference(set_B) = set_A - set_B`
- `set_A.intersection(set_B) = set_A & set_B`
- `set_A.issubset(set_B) = set_A <= set_B`
- `set_B.issuperset(set_A) = set_A <= set_A`
- `set_A.symmetric_difference(set_B) = set_A ^ set_B`

Hash value: `hash(object)` returns an integer that is unique to the object.
 Same hash values → same objects.
 This can be used for data lookup, data integrity checks and object uniqueness testing.



```
Last login: Fri Jun 30 15:31:50 on ttys004
python
Python 3.10.11 (main, May 17 2023, 14:30:36) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> set_a = set()
>>> set_a.
set_a.add(
set_a.clear(
set_a.copy(
set_a.difference(
set_a.difference_update(
set_a.discard(
set_a.intersection(
set_a.intersection_update(
set_a.isdisjoint(
>>> set_a.
```

Dictionary

A mapping object maps hashable values to arbitrary objects.

keys values

```
● ● ●
a = dict(one=1, two=2, three=3)

b = {'one': 1, 'two': 2, 'three': 3}

c = dict(zip(['one', 'two', 'three'], [1, 2,
3]))
d = dict([('two', 2), ('one', 1), ('three', 3)])

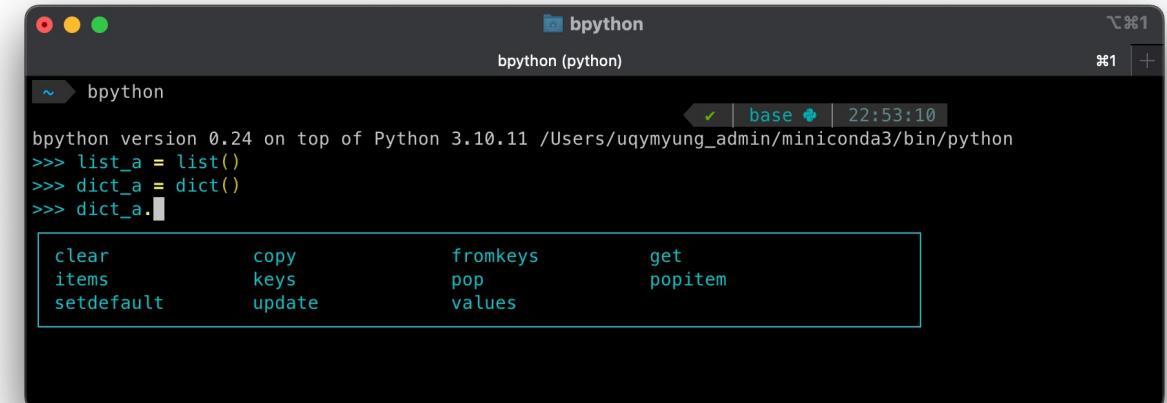
e = dict({'three': 3, 'one': 1, 'two': 2})

f = dict({'one': 1, 'three': 3}, two=2)

a == b == c == d == e == f
```

key value
{'ID-1': 'y.myung', 'ID-2': 'e.park', ...}

{'ID-1': {'name': 'Eric', 'age': 24}, ...}



```
bpython
bpython version 0.24 on top of Python 3.10.11 /Users/uqmyung_admin/miniconda3/bin/python
>>> list_a = list()
>>> dict_a = dict()
>>> dict_a.

clear        copy        fromkeys      get
items        keys        pop          popitem
setdefault   update      values
```

dict.keys(), dict.values() and dict.items() return the iterable of keys, values and items, respectively.

Q10. What happens if we add same keys?

Q11. What is difference between using slicing `[]` and `get()` method?

Compound statements

An essential part of the Python programming, which control the execution of other statements in some way.

[if, while, for, try, with, match, funcdef, classdef, async_with, async_for, async_func]

The `if` statement runs `<statement>` if the `<expr>` satisfies a condition.



```
x = -0.5
if 0 < x < 101:
    print(f"{x} is positive, but less than 101.")
elif x == 0:
    print(f"{x} is zero.")
elif x < 0:
    print(f"{x} is negative.")
else:
    print(f"{x} is greater than or equal to 101.")
```



Q12. Why the list 'd' is only identical to the list 'a'?



```
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> d
[1, 2, 3]
>>> a is b
False
>>> a is c
False
>>> a is d
True
```

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity (id)
is not	negate object identity (id)



```
if <expr>:
    <statement>
```

```
if <expr>:
    <statement>
else:
    <statement>
```

```
if <expr>:
    <statement>
elif <expr>:
    <statement>
```

```
if <expr>:
    <statement>
elif <expr>:
    <statement>
else:
    <statement>
```

Compound statements, if

An essential part of the Python programming, which control the execution of other statements in some way.

[if, while, for, try, with, match, funcdef, classdef, async_with, async_for, async_func]

The `if` statement runs `<statement>` if the `<expr>` satisfies a condition.

```
● ● ●
x = -0.5
if 0 < x < 101:
    print(f"{x} is positive, but less than 101.")
elif x == 0:
    print(f"{x} is zero.")
elif x < 0:
    print(f"{x} is negative.")
else:
    print(f"{x} is greater than or equal to 101.")
```

```
● ● ●
a = [1, 2, 3]
b = [1, 2, 3]
c = a.copy()
d = a
print(a is b)
print(a is c)
print(a is d)
print(id(a), id(b), id(c), id(d))
```

```
● ● ●
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> d
[1, 2, 3]
>>> a is b
False
>>> a is c
False
>>> a is d
True
>>> id(a),id(b),id(c),id(d)
(4382668928, 4370616960,
4370429248, 4382668928)
```

Operator	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity (id)
is not	negate object identity (id)

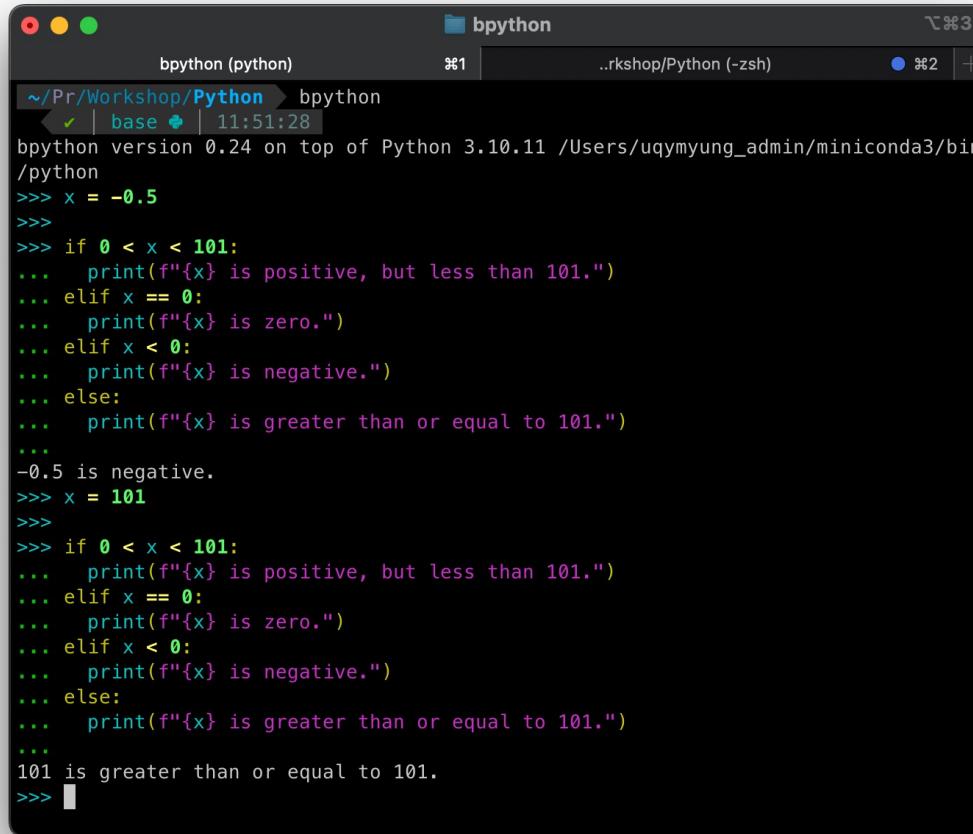
```
● ● ●
if <expr>:
    <statement>

if <expr>:
    <statement>
else:
    <statement>

if <expr>:
    <statement>
elif <expr>:
    <statement>

if <expr>:
    <statement>
elif <expr>:
    <statement>
else:
    <statement>
```

IF and Function



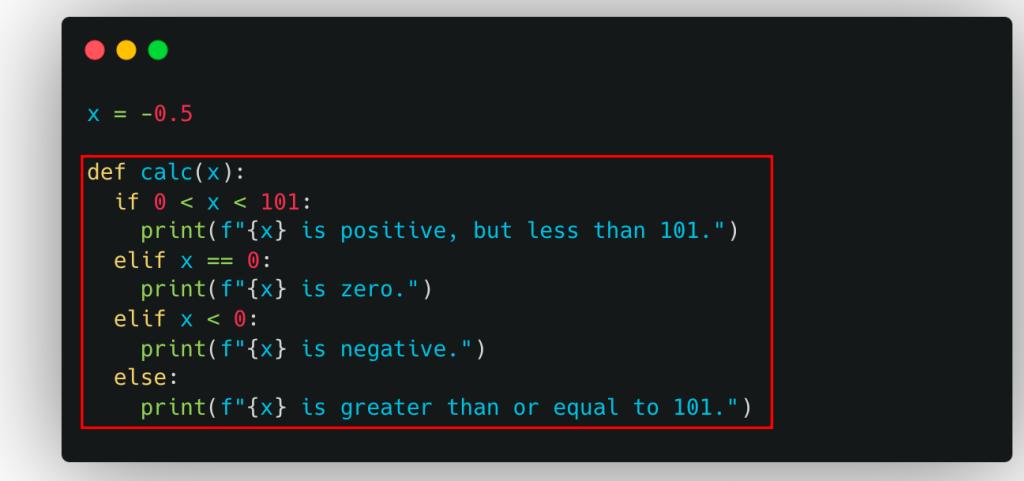
```

bpython (python)
~/.rkshop/Python (-zsh)  ☐  ✘  ☐  +  ↵⌘3
~/Pr/Workshop/Python > bpython
✓ | base ⚡ | 11:51:28
bpython version 0.24 on top of Python 3.10.11 /Users/uqymyung_admin/miniconda3/bin/python
>>> x = -0.5
>>>
>>> if 0 < x < 101:
...     print(f"{x} is positive, but less than 101.")
... elif x == 0:
...     print(f"{x} is zero.")
... elif x < 0:
...     print(f"{x} is negative.")
... else:
...     print(f"{x} is greater than or equal to 101.")

-0.5 is negative.
>>> x = 101
>>>
>>> if 0 < x < 101:
...     print(f"{x} is positive, but less than 101.")
... elif x == 0:
...     print(f"{x} is zero.")
... elif x < 0:
...     print(f"{x} is negative.")
... else:
...     print(f"{x} is greater than or equal to 101.")

101 is greater than or equal to 101.
>>> █

```



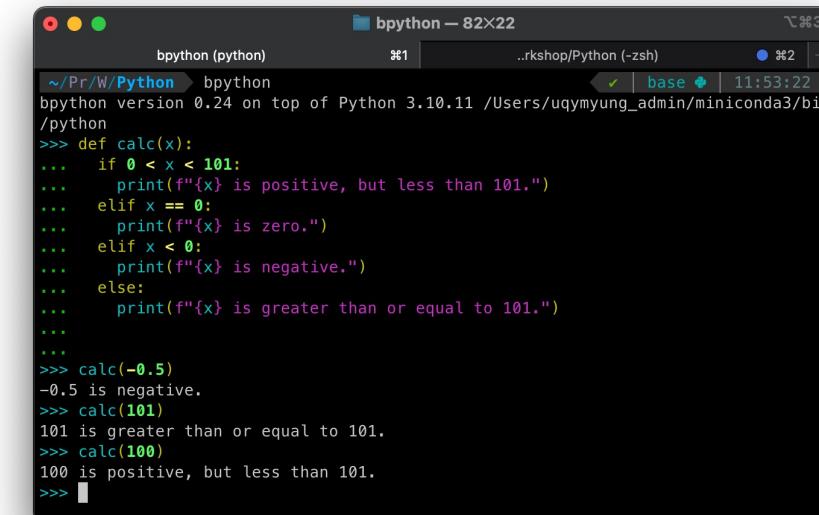
```

x = -0.5

def calc(x):
    if 0 < x < 101:
        print(f"{x} is positive, but less than 101.")
    elif x == 0:
        print(f"{x} is zero.")
    elif x < 0:
        print(f"{x} is negative.")
    else:
        print(f"{x} is greater than or equal to 101.")

█

```



```

bpython (python)
~/.rkshop/Python (-zsh)  ☐  ✘  ☐  +  ↵⌘3
~/Pr/W/Python > bpython
✓ | base ⚡ | 11:53:22
bpython version 0.24 on top of Python 3.10.11 /Users/uqymyung_admin/miniconda3/bin/python
>>> def calc(x):
...     if 0 < x < 101:
...         print(f"{x} is positive, but less than 101.")
...     elif x == 0:
...         print(f"{x} is zero.")
...     elif x < 0:
...         print(f"{x} is negative.")
...     else:
...         print(f"{x} is greater than or equal to 101.")

...
>>> calc(-0.5)
-0.5 is negative.
>>> calc(101)
101 is greater than or equal to 101.
>>> calc(100)
100 is positive, but less than 101.
>>> █

```

Function definitions

A function is an executable statement.

The function definition does not execute the function body; this gets executed only when the function is called.

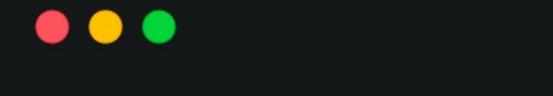
name parameter values

```
def multiplier(x1, x2):
    y = x1 * x2
    print(y)
    return y
```

4 spaces or tab

Indentation Levels:

Indentation is used to denote nested blocks of code, such as loops, conditionals, and function definitions. Each nested block should be indented by one-level deeper than its parent block.



```
y1 = multiplier(2, 3)
y2 = multiplier(3, 4)
y3 = multiplier(y1, y2)
```

Function to Class

A class is an executable statement,

```
● ● ●
def calculator(x1, x2):
    def multiplier(x1, x2):
        return x1 * x2
    def summation(x1, x2):
        return x1+x2
    def subtraction(x1, x2):
        return x1-x2

    return multiplier(x1,x2), summation(x1,x2), subtraction(x1,x2)
```

function

a **class** is a template for creating objects, while a **function** is a block of reusable code that performs a specific task

```
● ● ●
class calculator:
    def __init__(self):
        print("You initiated a calculator")
        self.x1 = 0
        self.x2 = 0
        self.history = list()

    def set_values(self, x1, x2):
        self.x1 = x1
        self.x2 = x2
        self.history.append([x1,x2])

    def multiplier(self):
        return self.x1 * self.x2
    def summation(self):
        return self.x1+self.x2
    def subtraction(self):
        return self.x1-self.x2

    def get_all_values(self):
        return self.multiplier(), self.summation(),
        self.subtraction()
```

constructor

method

FOR statement

The `for` statement is used to iterate over the elements of a sequence or other iterable object.

A `break` statement executed in the first suite terminates the loop without executing the `else` clause's suite.

A `continue` statement executed in the first suite skips the rest of the suite and continues with the next item.

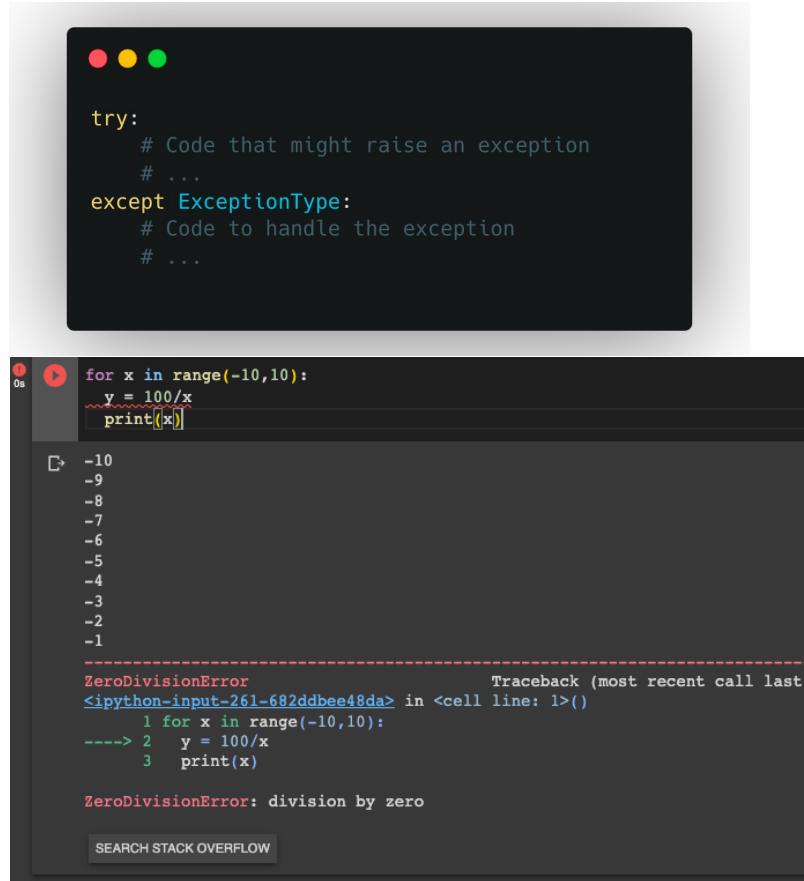
```
● ● ●  
for i in range(5):  
    print(i)  
  
for i in [0,1,2,3,4,5]:  
    print(i)  
  
for i in (0,1,2,3,4,5):  
    print(i)
```

Q13. Given the list below, how can we retrieve a new list that contains only words starting with the letter “A”?

```
● ● ●  
list_of_queries = ['ATC', 'ATT', 'TAC', 'ACT', 'AAA', 'ATA',  
'CTT', 'TTT', 'CAC']
```

TRY / EXCEPT statement

Try and Except are used to implement exception handling, which allows you to manage and respond to errors or exceptional conditions that may occur during the execution of a program.



```

try:
    # Code that might raise an exception
    # ...
except ExceptionType:
    # Code to handle the exception
    # ...

```

for x in range(-10,10):
 y = 100/x
 print(x)

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

ZeroDivisionError Traceback (most recent call last)
<ipython-input-261-682ddbee48da> in <cell line: 1>()
 1 for x in range(-10,10):
----> 2 y = 100/x
 3 print(x)

ZeroDivisionError: division by zero

[SEARCH STACK OVERFLOW](#)

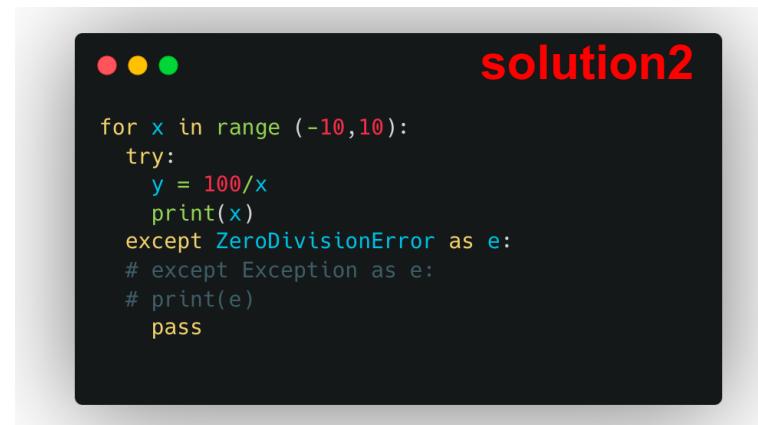


solution1

```

try:
    for x in range (-10,10):
        y = 100/x
        print(x)
except ZeroDivisionError as e:
# except Exception as e:
#     print(e)
    pass

```



solution2

```

for x in range (-10,10):
    try:
        y = 100/x
        print(x)
    except ZeroDivisionError as e:
#     except Exception as e:
#         print(e)
        pass

```

other statements

while: Indefinite iteration. Execute a statement repeatedly until a given condition is satisfied.



```
while <expr>:  
    <statement>
```

pass: The pass statement does nothing. It can be used when a statement is required syntactically but the program requires no action.



```
for x in range(1,10):  
    if x < 3:  
        print(f"{x} is less than 3")  
    elif x < 5:  
        print(f"{x} is less than 5")  
    else:  
        print(f"{x} is greater than or equal to 5")
```

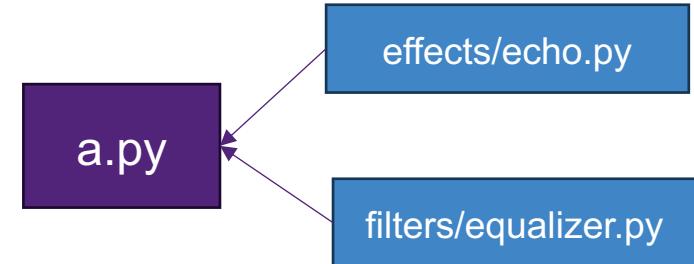


```
for x in range(1,10):  
    if x < 3:  
        print(f"{x} is less than 3")  
    elif x < 5:  
        pass  
    else:  
        print(f"{x} is greater than or equal to 5")
```

Modules and Packages

A module is a file containing Python definitions and statements.

The file name is the module name with the suffix .py appended.



Why we need to create a script?

> If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead.

```

sound/
    __init__.py           Top-level package
    formats/
        __init__.py       Subpackage for file format conversions
        wavread.py
        wavwrite.py
        aifhread.py
        aiffwrite.py
        auread.py
        auwrite.py
    ...
    effects/              Subpackage for sound effects
        __init__.py
        echo.py
        surround.py
        reverse.py
    ...
    filters/              Subpackage for filters
        __init__.py
        equalizer.py
        vocoder.py
        karaoke.py
        ...
  
```

sound/a.py

```

● ● ●

import sound.effects.echo
sound.effects.echo.a_echo("1000")

from sound.effects.echo import a_echo
a_echo("1000")

from sound.effects.reverse import real_reverse

a = real_reverse()
  
```

sound/effects/echo.py

```

● ● ●

def a_echo(a):
    print("from_a_echo",a)
  
```

sound/effects/reverse.py

```

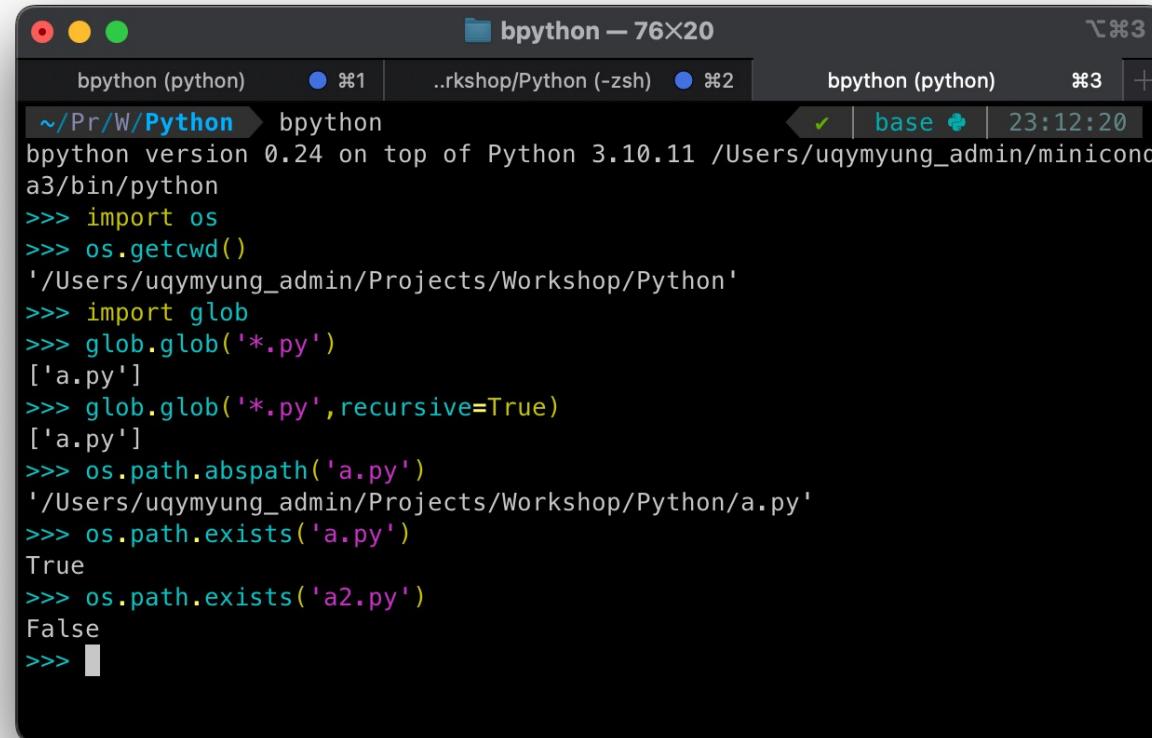
● ● ●

class real_reverse:
    def __init__(self):
        self.x = 1
        self.y = 2
    print("real_reverse was initiated.")
  
```

Standard Library

os: operating system interface, provides functions for interacting with the OS.

glob: provides a function for making file lists from directory searches.



```
~/Pr/W/Python bpython
bpython version 0.24 on top of Python 3.10.11 /Users/uqymyung_admin/minicond
a3/bin/python
>>> import os
>>> os.getcwd()
'/Users/uqymyung_admin/Projects/Workshop/Python'
>>> import glob
>>> glob.glob('*.*')
['a.py']
>>> glob.glob('*.*', recursive=True)
['a.py']
>>> os.path.abspath('a.py')
'/Users/uqymyung_admin/Projects/Workshop/Python/a.py'
>>> os.path.exists('a.py')
True
>>> os.path.exists('a2.py')
False
>>>
```

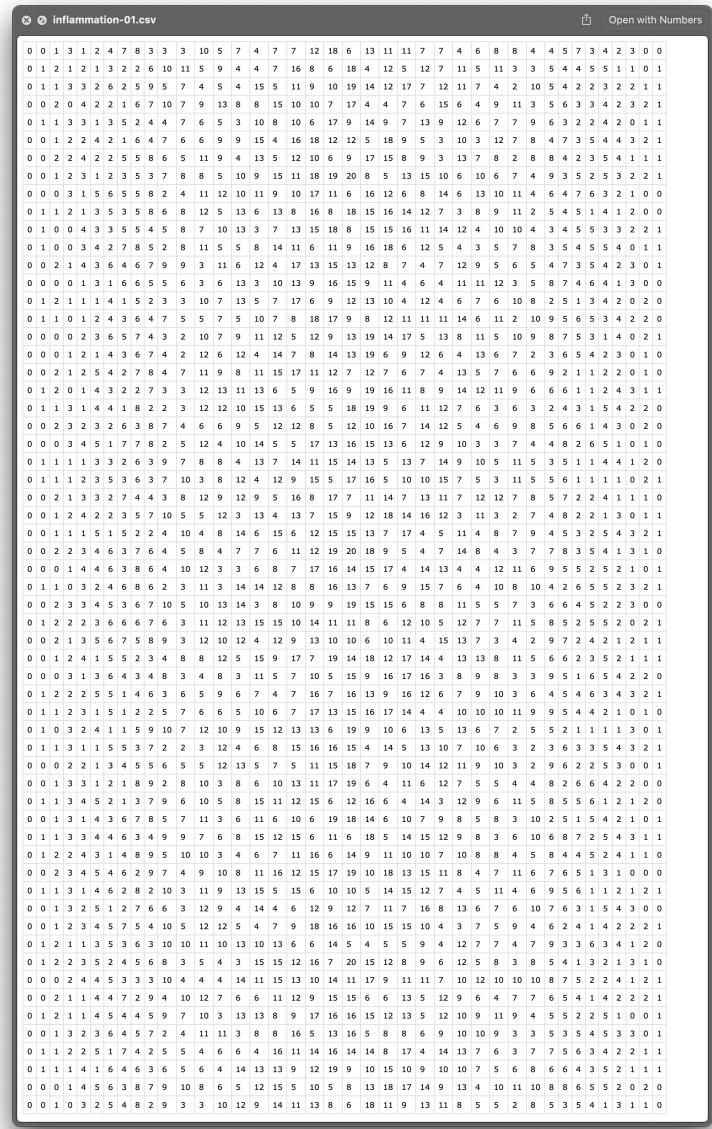
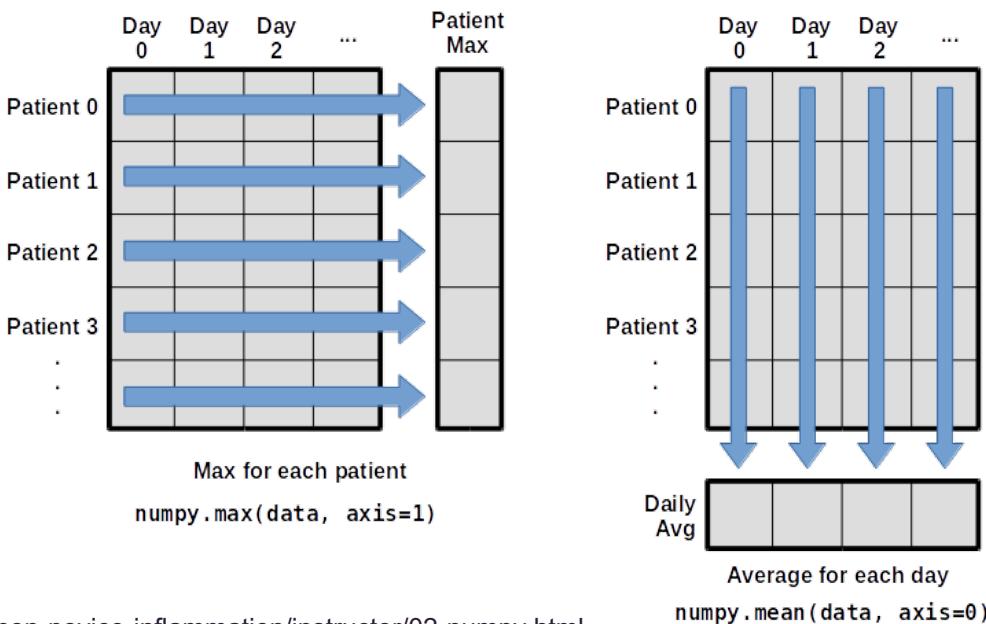
Analysing Patient Data



Requirements:

- a clinical trial inflammation data (csv); 60 arthritis patients for 40 days
 - a library called NumPy, Numerical Python, for slicing and analysing data

Q14. How to get “MIN/MAX/MEAN/STD” per patient or per day across all days or all patients?

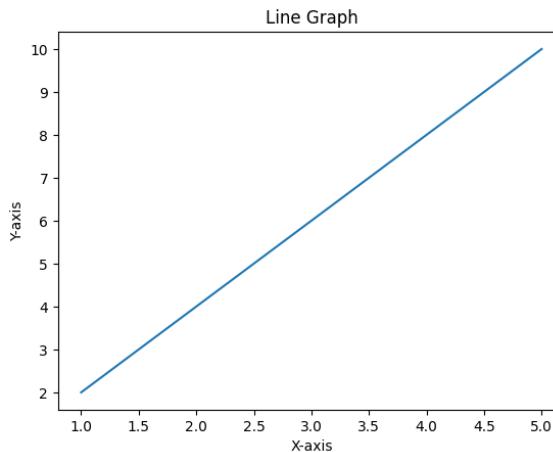


Data Visualisation - Matplotlib

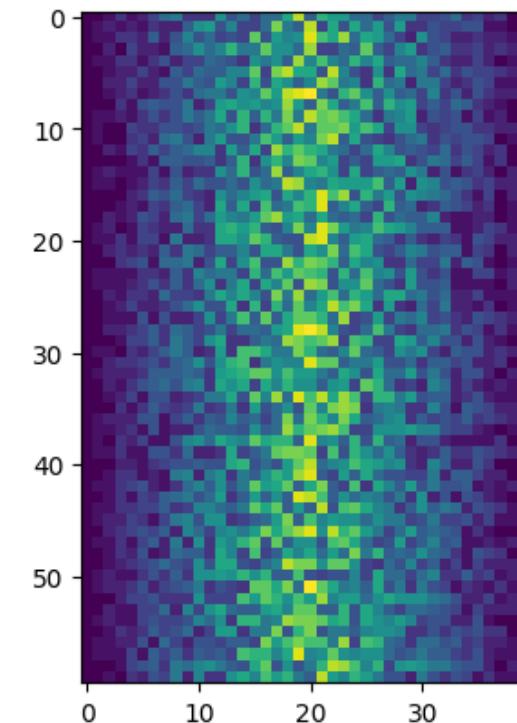
a popular Python library used for creating data visualisations.

- line plots, scatter plots, bar plots, histograms, pie charts and more.

```
● ● ●  
  
import matplotlib.pyplot as plt  
  
x = [1, 2, 3, 4, 5]  
y = [2, 4, 6, 8, 10]  
  
plt.plot(x, y)  
plt.xlabel('X-axis')  
plt.ylabel('Y-axis')  
plt.title('Line Graph')  
plt.show()
```



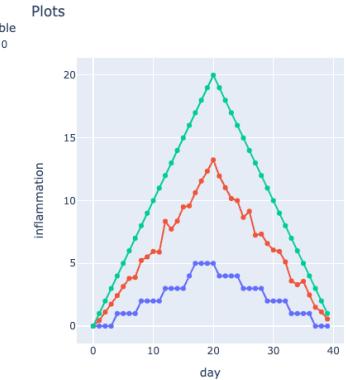
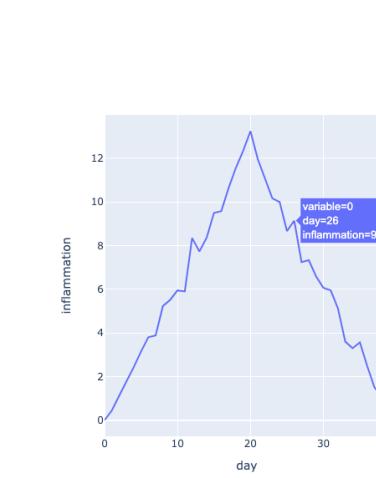
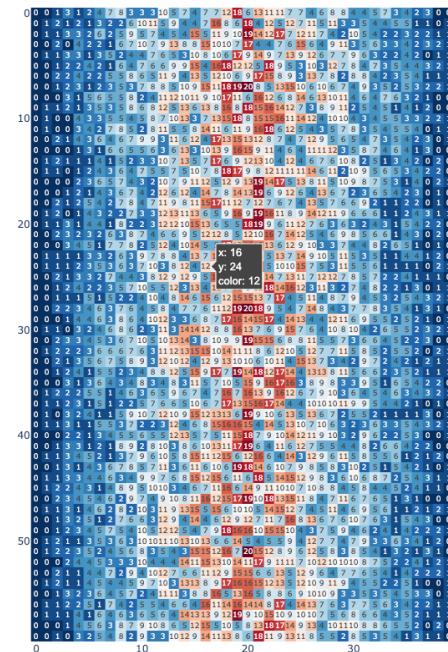
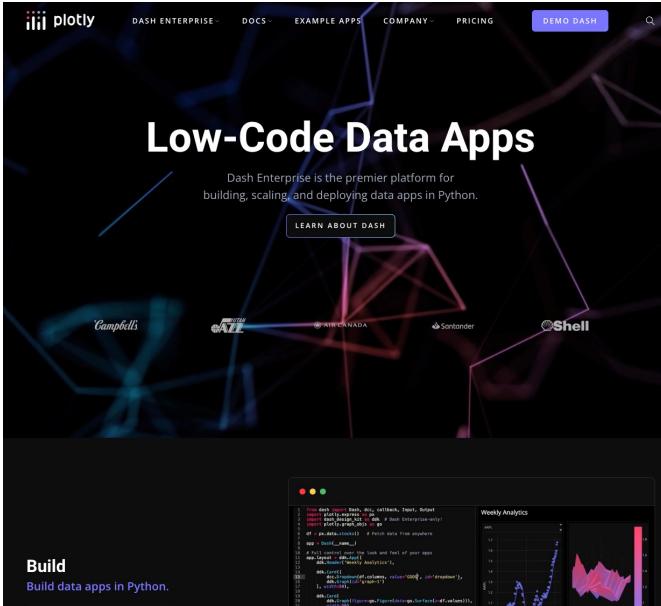
```
● ● ●  
  
image = plt.imshow(loaded_data)
```



Data Visualisation - Plotly



a Python library that allows for interactive data visualisation and provide wide range of graphing capabilities.



ChatGPT



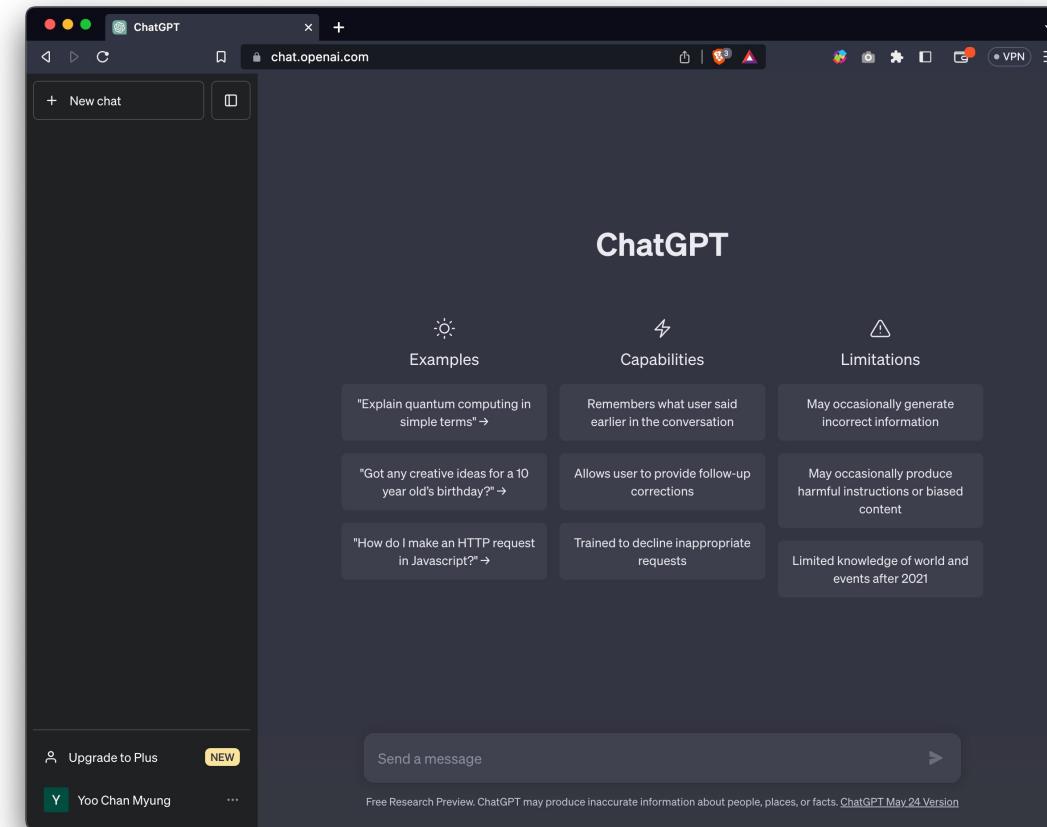
- ChatGPT is a language-based model designed for learning on sequential data.
- Due to the temperature parameter, ChatGPT offers flexibility and creativity, resulting in potentially different answers over time. (use “regenerate response”)

#1 Writing Code and Generating data

#2 Debugging Code

#3 Converting Code

You need an openAI account for using ChatGPT.



ChatGPT

Temperature give flexibility and creativity.

What is ChatGPT? (1st try)

ChatGPT is a language model developed by OpenAI. It stands for "Chat Generative Pre-trained Transformer." It is built upon the GPT (Generative Pre-trained Transformer) architecture, specifically GPT-3.5, which is designed to generate human-like text based on the given input. ChatGPT can understand and generate natural language responses, making it useful for a wide range of conversational tasks and applications. It can be used to answer questions, provide explanations, assist with writing, simulate dialogue, and more.

What is ChatGPT? (2nd try)

ChatGPT is a language model developed by OpenAI. It is part of the GPT (Generative Pre-trained Transformer) family of models and is based on GPT-3.5 architecture. GPT-3.5 is a highly advanced and powerful language model that has been trained on a large corpus of text data from the internet. It can generate human-like text and respond to prompts in a wide range of topics and languages.

ChatGPT can understand and generate natural language, making it capable of engaging in conversations, answering questions, generating code, providing explanations, and much more. It has been designed to be versatile and useful in various applications, including customer service, content generation, language translation, and more. Users can interact with ChatGPT through API calls, integrating it into their applications, or by accessing it through platforms like ChatGPT website or other interfaces.

ChatGPT: Writing Python code

Q. Create a heatmap using Plotly based on some data (random).

ChatGPT: Writing Python code

Q: Can you write **Python** code to plot a **heatmap** with **Plotly** using some **patient data**?

Y Can you write Python code to plot a heatmap with Plotly using some patient data?

Certainly! I can help you with that. To plot a heatmap using Plotly, we'll need patient data in a tabular format. For this example, let's assume we have patient data in a 2D array, where each row represents a patient and each column represents a feature or attribute of the patient. Here's an example code snippet that demonstrates how to create a heatmap using Plotly:

```
python
import plotly.graph_objects as go
import numpy as np

# Generate random patient data
np.random.seed(0)
patient_data = np.random.randn(10, 5) # 10 patients, 5 features

# Create the heatmap
fig = go.Figure(data=go.Heatmap(
    z=patient_data,
    colorscale='Viridis'))

# Set axis labels and title
fig.update_layout(
    xaxis=dict(title='Features'),
    yaxis=dict(title='Patients'),
    title='Patient Data Heatmap')

# Show the plot
fig.show()
```

In this example, we use NumPy to generate random patient data for demonstration purposes. You can replace `patient_data` with your actual patient data.

The `go.Heatmap` trace is used to create the heatmap, and we pass `z=patient_data` to provide the data for the heatmap. You can customize the colorscale to your liking by modifying the `colorscale` parameter.

The `update_layout` function is used to set the axis labels and title of the plot.

Finally, we use `fig.show()` to display the plot.

Please make sure you have Plotly installed (`pip install plotly`) and an active internet connection to see the generated heatmap.

brief explanation of the code



details about the code

ChatGPT: Writing Python code

Q: Can you create some patient data and apply to the above code?

Y Can you create some patient data and apply to the above code?

Certainly! Here's an example of creating patient data and applying it to the code snippet provided earlier:

```
python
import plotly.graph_objects as go
import numpy as np

# Generate patient data
patient_names = [ 'John', 'Alice', 'Bob', 'Emma', 'Michael' ]
attributes = [ 'Age', 'Weight', 'Height', 'Temperature' ]

patient_data = np.random.randint(18, 80, size=(len(patient_names), len(attributes)))
```

Create the heatmap
fig = go.Figure(data=go.Heatmap(
 z=patient_data,
 x=attributes,
 y=patient_names,
 colorscale='Viridis'))

Set axis labels and title
fig.update_layout(
 xaxis=dict(title='Attributes'),
 yaxis=dict(title='Patients'),
 title='Patient Data Heatmap')

Show the plot
fig.show()

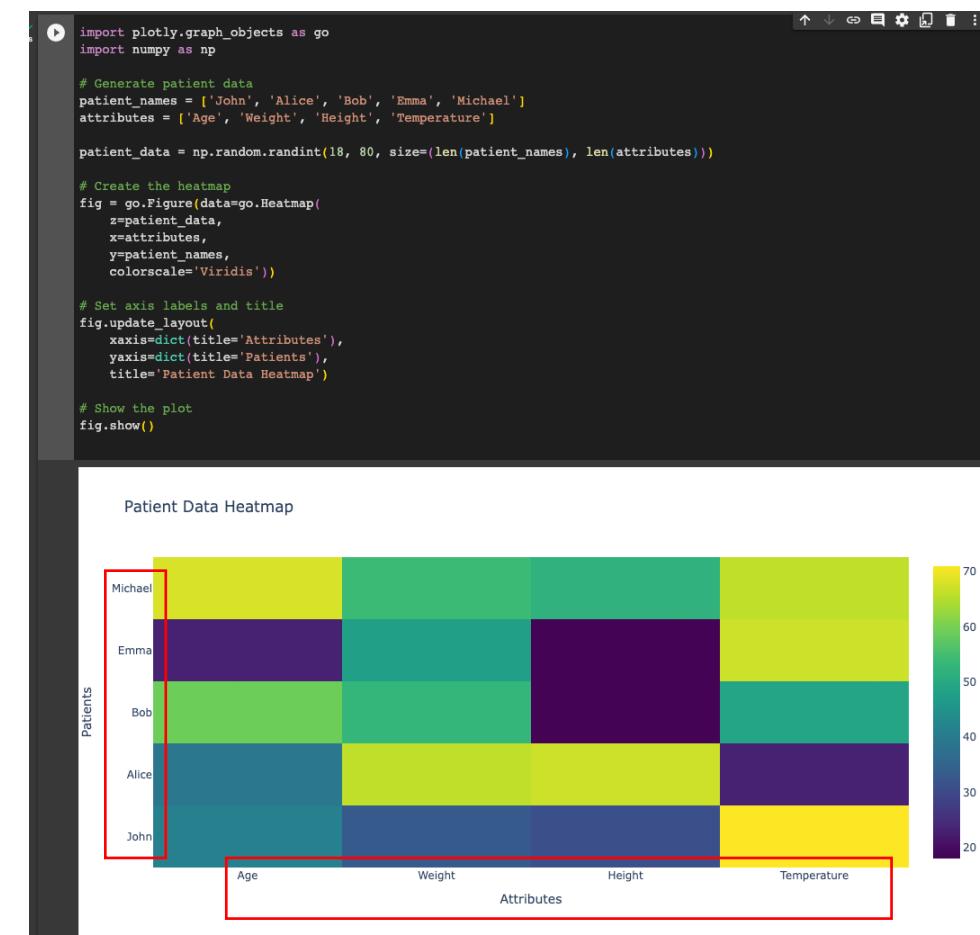
In this example, we define an array 'patient_names' containing the names of patients and an array 'attributes' containing the attributes or features to be plotted on the heatmap.

We generate random patient data using 'np.random.randint()', where each row represents a patient and each column represents an attribute. The values are randomly generated between 18 and 80 to represent patient age.

The 'go.Heatmap' trace is created with 'z=patient_data' for the data, 'x=attributes' for the x-axis labels, and 'y=patient_names' for the y-axis labels.

The remaining code is the same as before, setting the axis labels and title, and displaying the plot using 'fig.show()'.

Feel free to customize the patient names, attributes, and the range of data according to your specific needs.



ChatGPT: Writing Python code

Asking for Machine Learning examples

Y Can you write Python code to run machine learning on a dataset of 500 patients using the XGBoost classifier?

Certainly! I can provide you with an example of how to use the XGBoost classifier for machine learning on a dataset of 500 patients. Here's an example code snippet that demonstrates the steps involved:

```
python
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset (assuming it's in a CSV file)
dataset = pd.read_csv('patient_data.csv')

# Split the dataset into features and target variable
X = dataset.drop('target', axis=1)
y = dataset['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create an XGBoost classifier
model = xgb.XGBClassifier()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Y Write Python code for generating patient data and running machine learning with XGBoost on a dataset of 500 patients.

Certainly! Here's the Python code for generating patient data and running machine learning with XGBoost on a dataset of 500 patients:

```
python
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Set the number of patients and attributes
num_patients = 500
num_attributes = 5

# Generate random patient data
np.random.seed(0)
patient_data = np.random.randint(18, 80, size=(num_patients, num_attributes))

# Create column names for attributes
attribute_names = ['Age', 'Weight', 'Height', 'Temperature', 'Blood Pressure']

# Create a DataFrame from the patient data
df = pd.DataFrame(patient_data, columns=attribute_names)
df['Target'] = np.random.randint(0, 2, size=num_patients) # Adding a random target column

# Split the dataset into features and target variable
X = df.drop('Target', axis=1)
y = df['Target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create an XGBoost classifier
model = xgb.XGBClassifier()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

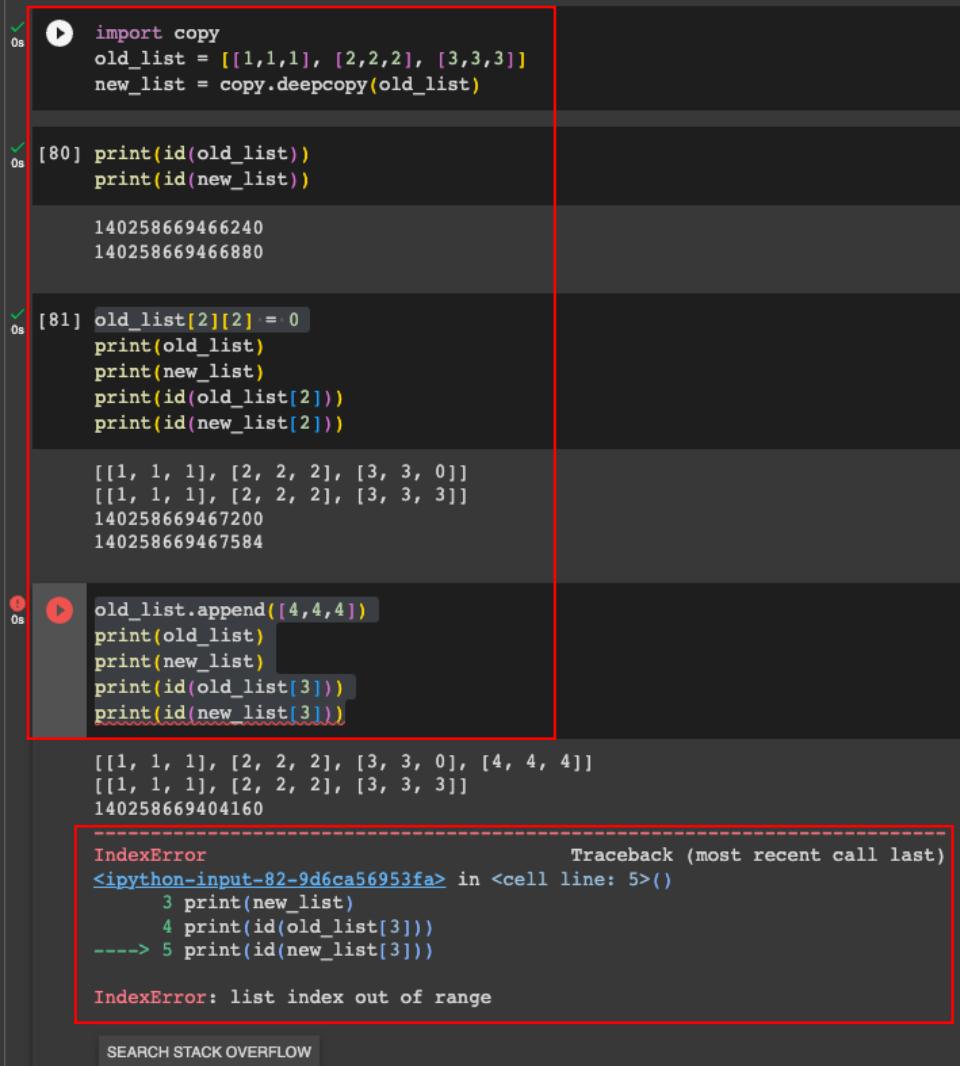
ChatGPT : Debugging

Y

Can you debug the following code?

```
import copy
old_list = [[1,1,1], [2,2,2], [3,3,3]]
new_list = copy.deepcopy(old_list)

old_list[2][2] = 0
old_list.append([4,4,4])
print(old_list)
print(new_list)
print(id(old_list[3]))
print(id(new_list[3]))
```

```
0s ⏪ import copy
old_list = [[1,1,1], [2,2,2], [3,3,3]]
new_list = copy.deepcopy(old_list)

0s [80] print(id(old_list))
print(id(new_list))

140258669466240
140258669466880

0s [81] old_list[2][2] = 0
print(old_list)
print(new_list)
print(id(old_list[2]))
print(id(new_list[2]))

[[1, 1, 1], [2, 2, 2], [3, 3, 0]]
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]
140258669467200
140258669467584

0s ⏴ old_list.append([4,4,4])
print(old_list)
print(new_list)
print(id(old_list[3]))
print(id(new_list[3]))

[[1, 1, 1], [2, 2, 2], [3, 3, 0], [4, 4, 4]]
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]
140258669404160

IndexError Traceback (most recent call last)
<ipython-input-82-9d6ca56953fa> in <cell line: 5>()
      3 print(new_list)
      4 print(id(old_list[3]))
----> 5 print(id(new_list[3]))

IndexError: list index out of range
```

SEARCH STACK OVERFLOW

ChatGPT : Debugging

Q: Can you debug the following code? (1st try)

The code you provided seems to be correct, and there is no apparent error in it. However, let's go through it and understand the behavior and output:

```
python
import copy

old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.deepcopy(old_list)

old_list[2][2] = 0
old_list.append([4, 4, 4])
print(old_list)
print(new_list)
print(id(old_list[3]))
print(id(new_list[3]))
```

This code demonstrates the usage of `copy.deepcopy()` to create a deep copy of `old_list` into `new_list`. Here's the breakdown of what the code does and the expected output:

1. `old_list` is a list of lists containing three sublists.
2. `new_list` is created using `copy.deepcopy(old_list)`, which creates a separate copy of `old_list` with independent nested lists.
3. The value at `old_list[2][2]` is modified to 0.
4. A new list `[4, 4, 4]` is appended to `old_list`.
5. `old_list` and `new_list` are printed.
6. The `id()` function is used to print the memory addresses of the fourth element in both lists.

The expected output of the code should be as follows:

```
lua
[[1, 1, 1], [2, 2, 2], [3, 3, 0], [4, 4, 4]]
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]
<id_of_old_list_element>
<id_of_new_list_element>
```

Note: The `<id_of_old_list_element>` and `<id_of_new_list_element>` values will be specific memory addresses and will differ from system to system.

If you are experiencing any specific issues or unexpected behavior, please let me know, and I'll be happy to assist you further.

ChatGPT : Debugging

Q: Can you debug the following code? (2nd try)

The code you provided appears to be correct and does not require any debugging. It performs a deep copy of the `old_list` using `copy.deepcopy()`, modifies the `old_list` by changing the value at `old_list[2][2]` to 0, and appends a new sublist `[4, 4, 4]` to `old_list`. After that, it prints both `old_list` and `new_list`, followed by printing the IDs of the fourth elements (`old_list[3]` and `new_list[3]`) using the `id()` function.

The output of the code should be as follows:

```
lua
[[1, 1, 1], [2, 2, 2], [3, 3, 0], [4, 4, 4]]
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]
[4, 4, 4]
[4, 4, 4]
```

The IDs of `old_list[3]` and `new_list[3]` will be the same because `copy.deepcopy()` creates a completely independent copy of the original list.

If you encounter any specific error or unexpected behavior, please provide more details, and I'll be happy to assist you further.

ChatGPT : Debugging

Y Are you sure?

 Apologies for the confusion. I made a mistake in my previous response. The code you provided does have an error.

Here's the corrected version of the code:

```
python
import copy

old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.deepcopy(old_list)

old_list[2][2] = 0
old_list.append([4, 4, 4])

print(old_list)
print(new_list)
print(id(old_list[3]))
print(id(new_list[2]))
```

In the last line of the code, the index for `new_list` should be `2` instead of `3`.

The corrected code will give the following output:

```
lua
[[1, 1, 1], [2, 2, 2], [3, 3, 0], [4, 4, 4]]
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]
140351722261888
140351722262016
```

Please let me know if you have any further questions or concerns.

Thank you

Yoochan Myung

Postdoctoral Research Fellow

y.myung@uq.edu.au



CRICOS 00025B • TEQSA PRV12080