Functions used in Python:

Beginner's Python Cheat Sheet

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

Lists (cont.)

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

Tuples

Tuples are similar to lists, but the items in a tuple can't be modified.

Making a tuple

```
dimensions = (1920, 1080)
```

If statements

If statements are used to test for particular conditions and respond appropriately.

Conditional tests

```
equals            x == 42
not equal         x != 42
greater than      x > 42
  or equal to     x >= 42
less than         x < 42
  or equal to     x <= 42
```

Conditional test with lists

```
'trek' in bikes
'surly' not in bikes
```

Assigning boolean values

```
game_active = True
can_edit = False
```

A simple if test

```
if age >= 18:
    print("You can vote!")
```

If-elif-else statements

```
if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

User input

Your programs can prompt the user for input. All input is stored as a string.

Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)

pi = input("What's the value of pi? ")
pi = float(pi)
```

1.

While loops

A while loop repeats a block of code as long as a certain condition is true.

A simple while loop

```
current_value = 1
while current_value <= 5:
    print(current_value)
    current_value += 1
```

Letting the user choose when to quit

```
msg = ''
while msg != 'quit':
    msg = input("What's your message? ")
    print(msg)
```

Functions

Functions are named blocks of code, designed to do one specific job. Information passed to a function is called an argument, and information received by a function is called a parameter.

A simple function

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

Passing an argument

```
def greet_user(username):
    """Display a personalized greeting."""
    print("Hello, " + username + "!")

greet_user('jesse')
```

Default values for parameters

```
def make_pizza(topping='bacon'):
    """Make a single-topping pizza."""
    print("Have a " + topping + " pizza!")

make_pizza()
make_pizza('pepperoni')
```

Defining a list

Use square brackets to define a list, and use commas to separate individual items in the list. Use plural names for lists, to make your code easier to read.

Making a list

```
users = ['val', 'bob', 'mia', 'ron', 'ned']
```

Accessing elements

Individual elements in a list are accessed according to their position, called the index. The index of the first element is 0, the index of the second element is 1, and so forth. Negative indices refer to items at the end of the list. To get a particular element, write the name of the list and then the index of the element in square brackets.

Getting the first element

```
first_user = users[0]
```

Getting the second element

```
second_user = users[1]
```

Getting the last element

```
newest_user = users[-1]
```

Modifying individual items

Once you've defined a list, you can change individual elements in the list. You do this by referring to the index of the item you want to modify.

Changing an element

```
users[0] = 'valerie'
users[-2] = 'ronald'
```

Importing Data — Python

```
pd.read_csv(filename) # From a CSV file
pd.read_table(filename) # From a delimited text file (like TSV)
pd.read_excel(filename) # From an Excel file
pd.read_sql(query, connection_object) # Reads from a SQL table/database
pd.read_json(json_string) # Reads from a JSON formatted string, URL or file.
pd.read_html(url) # Parses an html URL, string or file and extracts tables to a list of datafra
pd.read_clipboard() # Takes the contents of your clipboard and passes it to read_table()
pd.DataFrame(dict) # From a dict, keys for columns names, values for data as lists
```

Exploring Data — Python

```python
df.shape() # Prints number of rows and columns in dataframe
df.head(n) # Prints first n rows of the DataFrame
df.tail(n) # Prints last n rows of the DataFrame
df.info() # Index, Datatype and Memory information
df.describe() # Summary statistics for numerical columns
s.value_counts(dropna=False) # Views unique values and counts
df.apply(pd.Series.value_counts) # Unique values and counts for all columns
df.describe() # Summary statistics for numerical columns
df.mean() # Returns the mean of all columns
df.corr() # Returns the correlation between columns in a DataFrame
df.count() # Returns the number of non-null values in each DataFrame column
df.max() # Returns the highest value in each column
df.min() # Returns the lowest value in each column
df.median() # Returns the median of each column
df.std() # Returns the standard deviation of each column
```

Selecting Data — Python

```python
df[col] # Returns column with label col as Series
df[[col1, col2]] # Returns Columns as a new DataFrame
s.iloc[0] # Selection by position (selects first element)
s.loc[0] # Selection by index (selects element at index 0)
df.iloc[0,:] # First row
df.iloc[0,0] # First element of first column
```

Data Cleaning — Python

```python
df.columns = ['a','b','c'] # Renames columns
pd.isnull() # Checks for null Values, Returns Boolean Array
pd.notnull() # Opposite of s.isnull()
df.dropna() # Drops all rows that contain null values
df.dropna(axis=1) # Drops all columns that contain null values
df.dropna(axis=1,thresh=n) # Drops all rows have have less than n non null values
df.fillna(x) # Replaces all null values with x
s.fillna(s.mean()) # Replaces all null values with the mean (mean can be replaced with almost a
s.astype(float) # Converts the datatype of the series to float
s.replace(1,'one') # Replaces all values equal to 1 with 'one'
s.replace([1,3],['one','three']) # Replaces all 1 with 'one' and 3 with 'three'
df.rename(columns=lambda x: x + 1) # Mass renaming of columns
df.rename(columns={'old_name': 'new_ name'}) # Selective renaming
df.set_index('column_one') # Changes the index
df.rename(index=lambda x: x + 1) # Mass renaming of index
```

Filter, Sort, and Group By — Python

```python
df[df[col] > 0.5] # Rows where the col column is greater than 0.5
df[(df[col] > 0.5) & (df[col] < 0.7)] # Rows where 0.5 < col < 0.7
df.sort_values(col1) # Sorts values by col1 in ascending order
df.sort_values(col2,ascending=False) # Sorts values by col2 in descending order
df.sort_values([col1,col2], ascending=[True,False]) # Sorts values by col1 in ascending order t
df.groupby(col) # Returns a groupby object for values from one column
df.groupby([col1,col2]) # Returns a groupby object values from multiple columns
df.groupby(col1)[col2].mean() # Returns the mean of the values in col2, grouped by the values i
df.pivot_table(index=col1, values= col2,col3], aggfunc=mean) # Creates a pivot table that group
df.groupby(col1).agg(np.mean) # Finds the average across all columns for every unique column 1
df.apply(np.mean) # Applies a function across each column
df.apply(np.max, axis=1) # Applies a function across each row
```

Joining and Combining — Python

```python
df1.append(df2) # Adds the rows in df1 to the end of df2 (columns should be identical)
pd.concat([df1, df2],axis=1) # Adds the columns in df1 to the end of df2 (rows should be identi
df1.join(df2,on=col1,how='inner') # SQL-style joins the columns in df1 with the columns on df2
```

Writing Data — Python

```python
df.to_csv(filename) # Writes to a CSV file
df.to_excel(filename) # Writes to an Excel file
df.to_sql(table_name, connection_object) # Writes to a SQL table
df.to_json(filename) # Writes to a file in JSON format
df.to_html(filename) # Saves as an HTML table
df.to_clipboard() # Writes to the clipboard
```