Basic types

The following are the basic data/container types in Python.

str Plain text, surrounded by `''` or `""`. For example, `'string'` or `"string"`. (These are equivalent.)

int Integer value, a whole number without a decimal point. For example, `1`, `2`, `20`.

float Real number, a numerical value with a decimal point. For example `1.0`, `2.`, `20.0`, `20.5`.

list Simple container for objects, can be modified after initial creation. Items in the list can be accessed using their index. For example, `value = my_list[i]`. Append items to the end of a list using the `append` method: `my_list.append('item')`. Lists are created using the following pattern: `[1, 2,]`. (The trailing comma is advised.)

tuple Simple container for other objects, cannot be modified after initial creation. Items in the tuple can be accessed using their index. For example, i.e. `value = my_tuple[i]`. Tuples are created using the following pattern: `(1, 2,)`. (The trailing comma is advised.)

dict Dictionary container with key-based lookup. Items can be retrieved by providing the "key"; i.e. `value = my_dict['key']`. Dictionaries are created using the following pattern: `my_dict = {'k1': 1, 'k2': 2,}`.

Help, print, and dir

Three useful functions in Python for getting immediate help and debugging.

print Print the argument(s) to the terminal. Useful for inspecting the value of variables inside a function during runtime.

help Print the help (docstring) for a function, class, or module. Useful for quickly finding information about a function without leaving the terminal. For example, `help(map)` prints the help for the builtin `map` function.

dir Get the list of functions/classes in a module. Useful for quickly finding the name of a function from a module. Powerful when combined with the help function.

Functions

A function is defined as follows.

```
def name(arg1, arg2, arg_with='default'):
    """
    This is the docstring.
    Describe what the function does and
    the arguments it takes.
    """
    # body of the function goes here
    # Indentation signifies that this code
    # belongs to the function definition.
    return 'Return value'
```

This defines a function called `name`, which has two required parameters, `arg1` and `arg2`, and one optional parameter, `arg_with`, that has the default value `'default'`.

To call a function, use the name of the function followed by the arguments in brackets:

```
# arguments deduced from position
value = name('arg1', 'arg2')
# arg_with is given value 'arg3'
value = name('arg1', 'arg2', 'arg3')
# keywords deduced from keyword
value = name(arg1='arg1', arg2='arg2')
```

In al cases, the output (return value) is saved to the variable `value`.

Conditional statements

Python will interpret particular values as True or False in a conditional statment. For example, `None` is treated as False. Any non-empty list, tuple, or dict will be interpreted as True.

```
if condition:
    # code to execute if condition == True
    # Indent identifies code belonging to
    # the conditional.
```

For conditional statements require multiple alternatives, use elif and/or else.

```
if condition:
    # Do something when condition is True
elif other_condition:
    # Do something when other_condition is
      True
else:
    # Do something when the other conditions
       fail
```

Use the `not` keyword to negate the condition.

Loops and flow control

To loop over a range of integers, use a for loop:

```
for i in range(5):
    # for integers i = 0, 1, 2, 3, 4
    # Code to execute for each item goes
       here
    print(i) # print the item.
```

You can also loop over a Python list or tuple:

```
for item in my_list:
    print(item)
```

A while loop is implemented by the following.

```
while condition:
    # while condition is True
    # do the following.
```

The following keywords can be used in a loop:

`continue` Skip all remaining code in this iteration of the loop and start the next iteration of the loop.

`break` Leave the current loop, skipping any remaining code and iterations.

Opening files

To open a file, use the `open` function using a with statement:

```
with open('path/to/file', 'rt') as f:
    text = f.read()
```

(The path format above is for Linux/Mac, for Windows this will look different.) The mode by default is "read text" (`rb`). To open for writing use `wt`. The file object `f` can be written to by using the `write` method:

```
with open('path/to/file', 'wt') as f:
    f.write('some text\n')
    # \n is a new-line character
```

The with statement ensure that the file is properly closed even if an error occurs during the indented code.

Useful websites

`https://python.org`
The official Python website. Contains the documentation, guides, and links to many resources. The standard Python distribution can be downloaded here.

`https://pypi.org`
The Python Package Index (PyPI). A listing of Python packages that can be downloaded and installed using an installer tool such as pip.

`https://automatetheboringstuff.com`
Free e-book on practical uses of Python. Contains a comprehensive introduction to the language and lots of examples.

`https://realpython.org`
Website for *Pythonistas* (people who use Python). Contains many great tutorials.

`https://checkio.org`
Website for practising coding in Python by solving small problems. Great for quickly learning the basics. Starts basic but also have some tricky problems.

Common errors

The following are common errors. Note: the notes for each error are only a sample of common problems leading to each error. You may recieve these errors for other reasons.

SyntaxError
There is a problem with the Python code you have tried to run. Check the specified line to see if there is an obvious error, which may be an incorrect comma or missing bracket, or similar.

IndentationError
There is a line that should be indented that isn't, or a line that is indented that shouldn't be.

TypeError
A function has recieved an argument of a type that it cannot handle. Check the documentation for that function to see what type it expects for its input.

ValueError
A function has recieved an argument whose value is bad. For example, you have passed zero to the division function as the denominator argument.

IndexError
You have tried to retrieve an item from a list or tuple using an index that does not exist. Typically this means that the index provided is larger than the length of the list or tuple.

KeyError
You have tried to retrieve an item from a dictionary using a key that does not exist.

TabError
You have mixed tabs and spaces in formatting your source code. Indentation must be consistent. The recommended method is four space characters per indentation level.