Python Basics

The purpose of this document is not to make you a Python wizard, it is only to show the most basic syntax of the language in order for you to be able to make a basic program.

Variables

Variables are declared with only its name and value and are *not* declared with a type, Python will figure that out for you. There is also the special value of `None`, which is Python's null value.

Strings

Strings are variables containing letters. They have a length, they can be concatenated and they can be indexed.

Examples

```python
# A variable with a value of 10
foo = 10

# A variable containing a string
word = "foobar"

# A null variable
nothing = None

# A variable with a value of -3.14
neg_pi = -3.14

# Print the length of word
print(len(word))
# >>> 4

print(word + " forever")
# >>> foobar forever

print(word[0:3])
# >>> foo

# String formatting
print(f"Wow, I love {word}. 2 + 2 = {2 + 2}")
# >>> Wow, I love foobar. 2 + 2 = 4
```

Conditionals

The path of the program can be changed through the use of conditionals (if statements).

Examples

```
word = "foobar"
if len(foobar) == 5:
    print(f"{word} has a length of 5.")
elif len(foobar) == 6:
    print(f"{word} has a length of 6.")
else:
    print(f"{word} is not 5 or 6 characters in length.")
# >>> foobar has a length of 6

x = True
if x:
    print("x is Truthy")
else:
    print("x is Falsy")
# >>> x is Truthy
```

Loops

Python has two kinds of loops `for` loops and `while` loops. `for` loops enumerate over iterators, `while` loops loop until the condition becomes falsy.

Examples

```
for i in range(3):
    print(i)
# >>> 0
# >>> 1
# >>> 2
l = [1, 2, 3]
for x in l:
    print(x)
# >>> 1
# >>> 2
# >>> 3
x = True
while x:
    print(x)
    x = False
print(x)
```

```
# >>> True
# >>> False
```

Lists

Lists are collections that contain other variables in a specific order. Lists are heterogeneous, meaning the elements of the list can be of any type, and of mixed type.

Examples

```
# Creating an empty list
l = []

l = ['foo', 3.14, [True, -6], False]
print(l[1])
# >>> 3.14
print(l[2:])
# >>> [[True, -6], False]
l.append(42)
print(l)
# >>> ['foo', 3.14, [True, -6], False, 42]
l.push(99)
print(l)
# >>> [99, 'foo', 3.14, [True, -6], False, 42]
print(l.pop())
print(l)
# >>> 42
# >>> [99, 'foo', 3.14, [True, -6], False]
```

Dictionaries

Dictionaries (also know as hashmaps) are sets of key-value pairs. Both the keys and values are heterogeneous and can be of mixed types. The key-value pairs are stored in arbitrary order. The value is retrieved using the key.

Example

```
# Create and empty dictionary
d = {}
# Create a dictionary with 2 key-value pairs
d = {"foo": True, "bar": 3.14}

# Accessing a value
print(d["bar"])
# >>> 3.14
```

```python
# Adding a key-value pair
d['new'] = "Wowee, Python is cool"

# Iterate over key-value pairs
for k, v in d.items():
    print(f"Key: {k}, Value: {v}")
# >>> Key: foo, Value: True
# >>> Key: bar, Value: 3.14
# >>> Key: new, Value: Wowee, Python is cool
```

Functions

Functions divide your program into reusable, logical sections.

In Python, you don't need to declare the return type of a function, but you do need to declare the names of arguments that are passed to it.

Example

```python
# Declare a function
def func():
    # Do nothing
    pass

# Function with arguments
def func2(arg1, arg2):
    # Returning values
    return arg1 + arg2
```

Classes

Classes allow the grouping of data with functions.

Example

```python
class TestClass:

    # Constructor function
    def __init__(self, val):
        # Creating an instance variable
        self.instance_variable = val

    # Instance function
    def some_func(self, input_val):
        print(input_val)
        return input_val / 2
```

Files

File IO is one of the basic ways of getting input for a program.

Example

```
# Open a file for reading, and print each line
with open("file.txt", 'r') as f:
    for line in f:
        print(line)

# Create (overwrite) a new file and write to it
with open("file2.txt", "w") as f:
    f.write("foobar")
```

Main and Arguments

While python scripts can be run without a main function, it's best to run them with one. Getting command line arguments is also very useful, especially in programming competitions

Example

```
import sys

def main(args):
    print(f"Hello from {args[0]}")
    return 0

if __name__ == "__main__":
    main(sys.argv)
```