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

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

Hello world with a variable

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

Concatenation (combining strings)

```python
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

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

Get the first item in a list

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

Get the last item in a list

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

Looping through a list

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

Adding items to a list

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

Making numerical lists

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

Lists (cont.)

List comprehensions

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

Slicing a list

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

Copying a list

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

Tuples

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

Making a tuple

```python
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

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

Assigning boolean values

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

A simple if test

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

If-elif-else statements

```python
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

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

Accessing a value

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

Adding a new key-value pair

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

Looping through all key-value pairs

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

Looping through all keys

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

Looping through all the values

```python
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

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

Prompting for numerical input

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

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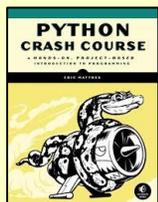
Python Crash Course

Covers Python 3 and Python 2

nostarchpress.com/pythoncrashcourse

While loops

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

A simple while loop

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

Letting the user choose when to quit

```python
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

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

greet_user()
```

Passing an argument

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

greet_user('jesse')
```

Default values for parameters

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

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

Returning a value

```python
def add_numbers(x, y):
    """Add two numbers and return the sum."""
    return x + y

sum = add_numbers(3, 5)
print(sum)
```

Classes

A class defines the behavior of an object and the kind of information an object can store. The information in a class is stored in attributes, and functions that belong to a class are called methods. A child class inherits the attributes and methods from its parent class.

Creating a dog class

```python
class Dog():
    """Represent a dog."""

    def __init__(self, name):
        """Initialize dog object."""
        self.name = name

    def sit(self):
        """Simulate sitting."""
        print(self.name + " is sitting.")

my_dog = Dog('Peso')

print(my_dog.name + " is a great dog!")
my_dog.sit()
```

Inheritance

```python
class SARDog(Dog):
    """Represent a search dog."""

    def __init__(self, name):
        """Initialize the sardog."""
        super().__init__(name)

    def search(self):
        """Simulate searching."""
        print(self.name + " is searching.")

my_dog = SARDog('Willie')

print(my_dog.name + " is a search dog.")
my_dog.sit()
my_dog.search()
```

Infinite Skills

If you had infinite programming skills, what would you build?

As you're learning to program, it's helpful to think about the real-world projects you'd like to create. It's a good habit to keep an "ideas" notebook that you can refer to whenever you want to start a new project. If you haven't done so already, take a few minutes and describe three projects you'd like to create.

Working with files

Your programs can read from files and write to files. Files are opened in read mode ('r') by default, but can also be opened in write mode ('w') and append mode ('a').

Reading a file and storing its lines

```python
filename = 'siddhartha.txt'
with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line)
```

Writing to a file

```python
filename = 'journal.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

Appending to a file

```python
filename = 'journal.txt'
with open(filename, 'a') as file_object:
    file_object.write("\nI love making games.")
```

Exceptions

Exceptions help you respond appropriately to errors that are likely to occur. You place code that might cause an error in the try block. Code that should run in response to an error goes in the except block. Code that should run only if the try block was successful goes in the else block.

Catching an exception

```python
prompt = "How many tickets do you need? "
num_tickets = input(prompt)

try:
    num_tickets = int(num_tickets)
except ValueError:
    print("Please try again.")
else:
    print("Your tickets are printing.")
```

Zen of Python

Simple is better than complex

If you have a choice between a simple and a complex solution, and both work, use the simple solution. Your code will be easier to maintain, and it will be easier for you and others to build on that code later on.

More cheat sheets available at
ehmatthes.github.io/pcc/