Beginner's Python Cheat Sheet

Variables and Strings
变量用于存储值。字符串是由单引号或双引号包围的一系列字符。

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
列表以特定的顺序存储一系列项。您可以使用索引或在循环中访问项。

创建一个列表

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

获取列表中的第一项

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

获取列表中的最后一项

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

遍历列表

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

向列表中添加项

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

创建一个数字列表

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

Lists (cont.)

列表推导式

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

切片列表

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

复制一个列表

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

Tuples
元组类似于列表，但是元组中的项不能被修改。

创建一个元组

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

If statements
*If*语句被用于检查特定的条件并返回特定的响应。

条件检查

```
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
```

条件检查是否在列表中

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

条件是否是布尔值

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

一个简单的If测试

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

If-elif-else 语句

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

Dictionaries
字典存储信息片段之间的联系。字典中的每一个项都是一个键值对

一个简单的字典

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

获取一个值

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

添加一个新的键值对

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

循环遍历所有的键值对

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

遍历所有键

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

遍历所有的值

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

User input
你的程序可以提示用户输入。所有输入都存储为字符串。

提示输入值

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

提示输入数值

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

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

While loops

*while*循环重复一个代码块，只要某个条件为真。

一个简单的while 循环

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

让用户选择何时退出

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

Functions

函数被命名为代码块，用于完成一项特定的工作。传递给函数的信息称为参数，函数接收到的信息称为参数。

一个简单的函数

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

greet_user()
```

传递一个参数

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

greet_user('jesse')
```

一个默认值参数

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

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

返回一个值

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

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

Classes

类定义了对象的行为和对象可以存储的信息类型。类中的信息存储在属性中，属于类的函数被称为方法。子类继承其父类的属性和方法。

创建一个 dog 类

```
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()
```

继承

```
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

如果你有无限的编程技能，你会做什么？

当你学习编程的时候，考虑一下你想要创建的真实项目是很有帮助的。记一个“想法”笔记本是个好习惯，你可以在任何时候开始一个新项目时查阅。如果你还没有这样做，花几分钟时间描述三个你想创建的项目。

Working with files

你的程序可以从文件中读取，也可以对文件进行写操作。默认情况下，文件以读模式('r')打开，但也可以以写模式('w')和附加模式('a')打开。

读取一个文件并存储它的行

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

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

写一个文件

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

附加一个文件

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

Exceptions

异常可以帮助您对可能发生的错误作出适当的响应。在try块中放置可能导致错误的代码。响应错误时应该运行的代码位于except块中。只有在try块成功时才应该运行的代码被放入else块。

捕获异常

```
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

简单胜过复杂

如果你要在简单和复杂的解决方案之间做出选择，并且两者都有效，那么就使用简单的解决方案。您的代码将更容易维护，而且您和其他人以后在该代码的基础上构建也将更容易。

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