Python Programming

Lecture 6 Functions, Modules

6.1 Functions Basics

Create functions

- A function is a named sequence of statements that performs a computation.
- Build-in Functions: int(), float(), str(), type(), and etc.
- To make your own functions, follow two steps: 1. define a function 2. call a function

```
def lyrics():
    print("I'm okay.")

lyrics()
```

- The first line of the function is called the header. The rest is called the body.

```
def repeat_lyrics():
    lyrics()
    lyrics()

repeat_lyrics()
```

```
I'm okay.
I'm okay.
```

Parameters and arguments (形参和实参)

```
1 def print_twice(bruce):
2     print(bruce)
3     print(bruce)
4 
5 print_twice('Spam')
6 print_twice(17)
```

```
Spam
Spam
17
17
```

```
1 michael = 'Eric, the half a bee.'
2 print_twice(michael)
```

```
Eric, the half a bee.
Eric, the half a bee.
```

Multiple parameters

- Positional Arguments

```
def describe_pet(type, name):
    print(f"My {type}'s name is {name}.")

describe_pet('hamster', 'Harry')
describe_pet('dog', 'Willie')
```

```
My hamster's name is Harry.
My dog's name is Willie.
```

```
describe_pet('harry', 'Hamster') #Order Matters
```

```
My harry's name is Hamster.
```

- Keyword Arguments (关键字参数)

```
def describe_pet(type, name):
    print(f"My {type}'s name is {name}.")

describe_pet(type='hamster', name='harry')
describe_pet(name='harry', type='hamster')
```

- Default Values

```
def describe_pet(name, type='dog'):
    print(f"My {type}'s name is {name}.")

describe_pet(name='willie')
```

```
My dog's name is Willie.
```

```
describe_pet(name='harry', type='hamster')
#The default value has been ignored.
```

- Example: Making an Argument Optional

```
def show_name(first, middle, last):
    full_name=first+' '+middle+' '+last
    print(full_name.title())
show_name('john','lee','hooker')
```

```
def show_name(first, last, middle=''):
    if middle:
        full_name=first+' '+middle+' '+last
    else:
        full_name=first+' '+last
    print(full_name.title())
show_name('jimi', 'hendrix')
```

Fruitful functions and void functions

- A fruitful function returns a value, while a void function performs an action but does not return a value.
- To return a result from a function, we use the return statement in our function.
- Return means the termination of a function. If you have two returns, only the first one will take effect.

```
1 def addtwo(a, b):
2     added = a + b
3     return added
4 x = addtwo(3, 5)
5 print(x)
```

```
8
```

```
1 def addtwo(a, b):
2     added = a + b
3
4 x = addtwo(3, 5)
5 print(x)
```

```
None
```

```
1 def addtwo(a, b):
2     added = a + b
3     print(added)
4
5 x = addtwo(3, 5)
6 print(x)
```

```
8
None
```

```
1 def addtwo(a, b):
2     added = a + b
3     print(added)
4     return added
5 x = addtwo(3, 5)
6 print(x)
```

```
8
8
```

- If we want to return multiple values, the function returns a tuple.

```
def addtwo(a, b):
    added = a + b
    return a, b, added

x = addtwo(3, 5)
print(x)
```

```
(3, 5, 8)
```

- A function can return any kind of value you need it to, including more complicated data structures like lists and dictionaries.

```
def build_person(first, last):
    person = {1: first, 2: last}
    return person

musician = build_person('jimi', 'hendrix')
print(musician)
```

```
{1: 'jimi', 2: 'hendrix'}
```

6.2 Modules（模块）

- Module is a Python file, followed with .py
- Storing Your Functions in Modules

```
def make_pizza(size):
    print(f"Making a {size}-inch pizza")
```

- It is saved as "pizza.py" file.
- We make a separate file called making_pizzas.py in the same directory as pizza.py.

```
import pizza

pizza.make_pizza(16)
```

Importing Specific Functions

```
#from module_name import function_name
#from module_name import function_0, function_1, function_2

from pizza import make_pizza

make_pizza(16)
```

Using as to Give a Function an Alias（别名）

```
from pizza import make_pizza as mp

mp(16)
```

Using as to Give a Module an Alias

```
import pizza as p

p.make_pizza(16)
```

- Importing All Functions in a Module

```
from pizza import *

make_pizza(16)
```

- The asterisk in the import statement tells Python to copy every function from the module pizza into this program file. Because every function is imported, you can call each function by name without using the dot notation.
- However, it's best not to use this approach when you're working with larger modules that you didn't write: if the module has a function name that matches an existing name in your project, you can get some unexpected results.

Search for Modules

- When a module named is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named xxx.py in a list of directories given by the variable sys.path. sys.path is initialized from these locations:
- The directory containing the input script (当前工作目录).
- PYTHONPATH (a list of directory names：标准库路径，第三方库路径)

```
1 import sys
2 print(sys.path)
```

Install third-party packages

- Anaconda already includes many useful packages, and you can install additional ones as needed.
- For Windows, open the anaconda prompt for anaconda, or open the cmd for original Python.

```
pip install pillow
#conda install pillow
```

- For macOS, open the terminal.

```
pip install pillow
#conda install pillow
```

- Uninstall the package

```
pip uninstall pillow
#conda uninstall pillow
```

- List all the packages

```
pip list
#conda list
```

Some third-party packages

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin','asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign',
'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot',
'inf', 'isclose', 'isfinite', 'isinf','isnan', 'isqrt', 'lcm', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter','perm',
'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

```
import math

print(math.pi)
print(math.exp(50))
print(math.sin(math.pi/2))
print(math.acos(0.5))
print(math.log(5, 20))
```

```python
import random

print(random.randint(1, 10))          # 产生 1 到 10 的一个整数型随机数
print(random.random())                # 产生 0 到 1 之间的随机浮点数
print(random.uniform(1.1, 5.4))       # 产生  1.1 到 5.4 之间的随机浮点数
print(random.choice('tomorrow'))      # 从序列中随机选取一个元素
print(random.randrange(1, 100, 2))    # 生成从1到100的间隔为2的随机整数
a = [1, 3, 5, 6, 7]
random.shuffle(a)                     # 将序列a中的元素顺序打乱
print(a)
```

you-get安装使用说明

```
pip install you-get
# youtube-dl
```

Free Python Games官网

```
pip install freegames
```

6.3 Blackjack (21点游戏)

21点游戏规则

- 玩法：玩家与电脑比拼手牌点数，接近 21 且不超越者获胜。
- 玩家的目标是让手中的牌点数尽可能接近21点，但不能超过21点，否则称为“爆牌”，立即输掉本局。
  - A（Ace，A牌）可以是1点或11点，由玩家决定（如A和6可算作7或17）
  - 2-10点按牌面值计算。J（Jack）、Q（Queen）、K（King）均为10点。
- 游戏流程
  - 1. 发牌：玩家和庄家各获得两张牌
  - 2. 玩家回合：要牌（再抽一张牌）或者，停牌（不再要牌）。
  - 3. 庄家回合：要牌，或者，停牌，庄家的牌点数小于17则必须要牌。
  - 4. 胜负判断：
    - 玩家点数 > 21：玩家输。
    - 玩家点数 ≤ 21，且比庄家高，或庄家爆牌：玩家赢。
    - 玩家和庄家点数相同：平局。

教学视频: 赌棍 The Gambler (2014)

```python
import random

# 2-10直接用数值表示，JQK作为10，A作为11（可以调整为1）
def create_deck():
    values = [2,3,4,5,6,7,8,9,10,10,10,10,11]
    deck = values * 4  # 4种花色
    random.shuffle(deck)
    return deck

# 计算手牌点数（考虑 A=11 变 A=1 的情况）
def calculate(cards):
    total = sum(cards)
    aces = cards.count(11) # 数A个数
    while total > 21 and aces > 0:
        total -= 10  # 将A从11变1
        aces -= 1
    return total
```

```python
# 玩家回合
def player_turn(player_hand, deck):
    while True:
        score = calculate(player_hand)  # 调用计算函数
        print(f"玩家手牌: {player_hand}, 当前点数: {score}")
        if score >= 21:
            break
        move = input("是否要牌? (y/n): ")
        if move == 'y':
            player_hand.append(deck.pop())
        else:
            break
    return score, deck  # 直接返回计算出的分数

# 庄家回合（电脑回合），只要小于17就继续要牌
def dealer_turn(dealer_hand, deck):
    while calculate(dealer_hand) < 17:
        dealer_hand.append(deck.pop())
    return calculate(dealer_hand), deck
```

```python
def blackjack():
    deck = create_deck()
    # 发两张初始手牌
    player_hand = [deck.pop(), deck.pop()]
    dealer_hand = [deck.pop(), deck.pop()]
    # 玩家回合
    player_score, deck = player_turn(player_hand, deck)
    if player_score > 21:
        print("玩家爆牌，庄家获胜！")
        return
    # 庄家回合
    dealer_score, deck = dealer_turn(dealer_hand, deck)
    print(f"庄家手牌: {dealer_hand}，点数: {dealer_score}")
    # 判定胜负
    if dealer_score > 21 or player_score > dealer_score:
        print("玩家获胜！")
    elif player_score == dealer_score:
        print("平局！")
    else:
        print("庄家获胜！")
blackjack()
```

Summary

- Functions
  - Reading: Python for Everybody, Chapter 4
  - Reading: Python Crash Course, Chapter 8