Week1

`CS 61A 2021 Fall` 官网：CS 61A: Structure and Interpretation of Computer Programs
`翻译视频`：【计算机程序的构造和解释】精译【UC Berkeley 公开课-CS61A (Spring 2021)】-中英双语字幕
`github`:Maxwell2020152049/CS61A

Wed 1/20 Lecture #1: Introduction

`Lab`： Lab 00: Getting Started
`Slide`： 01-Introduction_full.pdf

Computer Science

简单介绍了计算机科学的分支：

- `Programming`
- `Theory`
- `Programming Languages`
- `Graphics`
- `Artificial Intelligence`
- `Systems`

`CS 61A` 是关于 `Programming` (编程)的课程，需要使用一门编程语言 `Python` 。

CS10

如果感觉本课程太困难，可以考虑 `CS10` ，关于 `CS10`： https://cs10.org/su23/

Course Organization

- `Readings`
- `Lectures`
- `Labs`
- `Discussion`
- `Tutorials`
- `Homework`
- `Projects`

Fri 1/22 Lecture #2: Functions, Expressions, Environments

`Homework`： Homework 1: Variables & Functions, Control
`Slide`： 02-Functions_full.pdf

Functions as Values

在 `Python` 中， `functions` (函数)可以作为 `fisrt class values` （第一类实体）

> 在计算机科学中，"first-class values" 是指将值（或数据）视为第一类实体，具有与其他实体相同的地位和权利。这意味着在编程语言中，可以像对待其他变量、函数或对象一样对待值。
> 首先，"first-class values" 可以存储在变量中，就像任何其他值一样。可以将它们传递给函数作为参数，也可以从函数中返回。这使得值可以在程序中被传递和操作，而不仅限于某些特定的用法。

> 其次，"first-class values" 可以赋予其他特性，例如可以动态地创建和销毁。这允许在运行时生成新的值，以便根据需要进行计算和处理。
> 最重要的是，"first-class values" 具有与其他语言构造（如函数、方法、类等）相同的能力和权益。这包括对值进行操作、组合、扩展和抽象的能力，从而使程序具有更高的灵活性和表达能力。
> 使用支持 "first-class values" 的编程语言，开发者可以更自由地处理数据，并以更精确的方式描述问题和解决方案。这种概念在函数式编程、面向对象编程和许多现代编程语言中都得到了广泛应用。

Functions Values

在 `Python` 中，函数分为两种：

- `Python` 的内置函数（`built-in Function`）

```
func abs(number), func add(left, right)
```

- 用户自定义函数（`function definitions`）

```
def saxb(a, x, b): # Header: Name and formal parameters
    return a * x + b # Body: Computation performed by function
```

函数的参数列表是函数签名的一部分。

Anonymous Functions（匿名函数）

在 `Python` 中，可以使用 `lambda` 表达式定义匿名函数，上述 `saxb()` 函数可以定义如下：

```
lambda a, x, b: a * x + b
```

Pure Functions（纯函数）

纯函数调用参数列表传入的实参，并进行计算，返回结果，没有副作用（`side effects`）。

Impure Functions（非纯函数）

非纯函数除了返回值之外，还会产生副作用（`side effects`），像 `print()` 函数，会有输出（`output`）；随机值函数 `randint()` 也有副作用。

Call Expressions

> A call expression denotes the operation of calling a function.

调用顺序是 `从外到内，从左到右`。
下面的一个调用表达式很好地展示了这个原则（`2` 在被调用前是 `numeral`，`2` 在被调用前是 `number`）：

> "Numeral"和"number"是两个与数字相关的术语，它们有一些区别：
>
> 1. Number（数字）：Number是一个广义的术语，用于表示数值或数量。它可以指代任何实际的、抽象的或符号化的数值，无论是整数、分数、小数还是负数。Number是数学中基本的概念，并且通常用来进行计算、比较和量化。
> 2. Numeral（数字符号）：Numeral是一种表示数字的符号或字符。它是一种书写系统，用于表示具体的数值。不同的文化和语言使用不同的数字符号和系统。例如，阿拉伯数字系统（0、1、2、3等）是世界上最常见的数字符号，而罗马数字（I、V、X、L等）是罗马帝国时期广泛使用的一种特殊符号系统。
>
> 总结起来，"number"是指任何数值或数量，而"numeral"是指用于表示具体数值的符号或字符

```
from operator import mul, add
mul(add(2, mul(0x10, 0o10)), add(0x3, 5))
```

思考下面 `Python` 语句的输出：

```
print(print(1), print(2))
```

函数名也是函数签名的一部分。

Substitution

`Python` 定义变量名的方法：

- `assignment`
- `functions definitions`
- `parameter passing`

简单的代换示例1：

```
def compute(x, y):
    return (x * y) ** x
```

```
>>> compute(3, 2)
216
```

简单的代换示例2：

```
def incr(n):
    def f(x):
        return n + x
return f
```

```
>>> incr(5)(6)
11
```

简单的代换示例3（ `f()` 内部的 `x` 不会被）：

```
def hmmmm(x):
    def f(x):
        return x
    return f
```

```
>>> hmmmm(5)(6)
6
```

对下面这个例子，代换就不能很好的解释了：

```
x = 4
x = 8
```

```
print(x)
8
```

这时候，就需要引入环境（ `environment` ）来进行解释。

Environment

> - An environment is a mapping from names to values.
> - We say that a name is bound to a value in this environment.
> - In its simplest form, it consists of a single global environment frame:

Evaluation

> - Every expression is evaluated in an environment, which supplies the meanings of any names in it.
> - Evaluating an expression typically involves first evaluating its subexpressions (the operators and operands of calls, the operands of conventional expressions such as `x*(y+z)` , . . .).
> - These subexpressions are evaluated in the same environment as the expression that contains them.
> - Once their subexpressions (operator + operands) are evaluated, calls to user-defined functions must evaluate the expressions and statements from the definitions of those functions.

`Environment` 和 `Evaluation` 这个概念可以解释下面的程序:

```
from operator import mul
def square(x):
   return mul(x,x)
x = -2
```

```
>>> square(mul(x, x))
16
```