# Review:
# Lab03 & Hw03

# 今日实验课内容

这里有页码
提问请指出第几页

- 作业题选讲:
  - lab03p5: Maximum Subsequence
  - hw03p2: Ping-pong
  - hw03p4: Count Change
  - hw03p6: Multiadder
  - hw03p8: All-Ys Has Been
- 一些学习方面的建议


不想听的话可以直接开搞lab04了

# lab03p5: Maximum Subsequence

- max_subseq(n, l): 求n的长度不超过l子序列构成的最大的数

max_subseq(20125, 3) = 225

- Base cases:

# lab03p5: Maximum Subsequence

- max_subseq(n, l)：求n的长度不超过l子序列构成的最大的数

max_subseq(20125, 3) = 225

- Base cases:
  - n == 0：数字是0，结果只可能是0
  - l == 0：长度为0的子序列只可能是0
  - 注意: 不要选n < 10或者l == 1作为base case，会变得很复杂

# lab03p5: Maximum Subsequence

- max_subseq(n, l)：求n的长度不超过l子序列构成的最大的数

max_subseq(20125, 3) = 225

- n > 0 and l > 0?
  - 只需要考虑末尾一位在不在答案里就够了
  - 不在答案：return max_subseq(n // 10, l)
  - 在答案里：

# lab03p5: Maximum Subsequence

- max_subseq(n, l)：求n的长度不超过l子序列构成的最大的数

max_subseq(20125, 3) = 225

- n > 0 and l > 0?
  - 只需要考虑末尾一位在不在答案里就够了
  - 不在答案：return max_subseq(n // 10, l)
  - 在答案里：return max_subseq(n // 10, l - 1) * 10 + (n % 10)

# lab03p5: Maximum Subsequence

- max_subseq(n, l): 求n的长度不超过l子序列构成的最大的数

max_subseq(20125, 3) = 225

- n > 0 and l > 0?

```
return max(max_subseq(n // 10, l),
           max_subseq(n // 10, l - 1) * 10 + (n % 10))
```

# hw03p2: Ping-pong

- 求Pingpong数列的第n项

- 用while写： SO EASY!
- 用递归写：

# hw03p2: Ping-pong

- 求Pingpong数列的第n项

- 一种错误的想法：

  - Base case: n == 1 => 1
  - Recursion: pingpong(n) = pingpong(n-1) +1/-1

  - 整了半天，做不出来，或者程序跑得贼慢 

# hw03p2: Ping-pong

- 求Pingpong数列的第n项

- 换个思路：先写while的做法，然后改成递归（由莉莉丝提供代码）

```python
def pingpong(n):
    curr, result, direct = 1, 1, 1
    while curr != n:
        if curr % 6 == 0 or number_of_six(curr) > 0:
            result = result - direct
            direct = -direct
        else:
            result = result + direct
            # direct = direct
        curr = curr + 1
    return result
```

# hw03p2: Ping-pong

- 求Pingpong数列的第n项
- 换个思路：先写while的做法，然后改成递归（由莉莉丝提供代码）

```python
def pingpong(n):
    def helper(curr, result, direct):
        if curr == n:
            return result
        if curr % 6 == 0 or number_of_six(curr) > 0:
            return helper(curr + 1, result - direct, -direct)
        return helper(curr + 1, result + direct, direct)
    return helper(1, 1, 1)
```

# hw03p2: Ping-pong

- 求Pingpong数列的第n项

- 再换个思路：

  先确定n的时候的方向
  然后递归求解
  (由莉莉丝提供代码)

```python
def pingpong(n):
    def ping_pong(k, f):
        if k <= 6:
            return k
        elif k % 6 == 0 or number_of_six(k) > 0:
            return ping_pong(k - 1, -f) - f
        else:
            return ping_pong(k - 1, f) + f
    def add_or_sub(k):
        if k < 6:
            return 1
        elif k % 6 == 0 or number_of_six(k) > 0:
            return -1 * add_or_sub(k - 1)
        else:
            return add_or_sub(k - 1)
    return ping_pong(n, add_or_sub(n))
```

# hw03p2: Ping-pong

- 求Pingpong数列的第n项
- 再换个思路：pingpong(k)和pingpong(n)相差多少呢？

# hw03p2: Ping-pong

- 求Pingpong数列的第n项
- 再换个思路： pingpong(k)和pingpong(n)相差多少呢?
  - Base case: k == n => 

# hw03p2: Ping-pong

- 求Pingpong数列的第n项
- 再换个思路： pingpong(k)和pingpong(n)相差多少呢？
  - Base case: k == n => 0
  - Recursion: diff(k) =

# hw03p2: Ping-pong

- 求Pingpong数列的第n项

- 再换个思路： pingpong(k)和pingpong(n)相差多少呢?

  - Base case: k == n => 0
  - Recursion: diff(k) = pingpong(n) – pingpong(k)
    = pingpong(n) – (pingpong(k+1) - (1/-1))
    = diff(k + 1) + (1/-1)

# hw03p2: Ping-pong

- 求Pingpong数列的第n项

- 再换个思路： pingpong(k)和pingpong(n)相差多少呢?
  - Base case: k == n => 0
  - Recursion: diff(k) = diff(k + 1) + (1/-1)

加1还是-1怎么算?

# hw03p2: Ping-pong

- 求Pingpong数列的第n项

- 再换个思路： pingpong(k)和pingpong(n)相差多少呢?

  - Base case: k == n => 0
  - Recursion: diff(k, d) = diff(k + 1, -d if … else d) + d

  - Answer: pingpong(n) = diff(0, 1)

# hw03p4: Count Change

- count_change(total, next_money): 求找零的方法数量

- Base case:
  - total < 0 =>
  - total == 0 =>
  - money is None =>

# hw03p4: Count Change

- count_change(total, next_money): 求找零的方法数量

- Base case:
  - total < 0 => 0
  - total == 0 => 1
  - money is None => 0
- Recursion:
  - 多用一张当前的钱：count(total – money, money)
  - 用下个更大的面额：count(total, next_money(money))

# hw03p6: Multiadder

- **multiadder(n)**：返回一个可以连续调用**n**次的求和函数

# hw03p6: Multiadder

- multiadder(n)：返回一个可以连续调用n次的求和函数
- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

# hw03p6: Multiadder

- multiadder(n): 返回一个可以连续调用n次的求和函数
- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn
- Base case: 

# hw03p6: Multiadder

- multiadder(n): 返回一个可以连续调用n次的求和函数
- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn
- Base case: n == 1 => lambda x: x

# hw03p6: Multiadder

- multiadder(n): 返回一个可以连续调用n次的求和函数

- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

- Base case: n == 1 => lambda x: x
- Recursion:

# hw03p6: Multiadder

- multiadder(n): 返回一个可以连续调用n次的求和函数
- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

lambda x12: lambda x3: …: lambda xn: x12 + x3 + … + xn

# hw03p6: Multiadder

- multiadder(n)：返回一个可以连续调用n次的求和函数
- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

lambda x12: lambda x3: …: lambda xn: x12 + x3 + … + xn

lambda x123: lambda x4: …: lambda xn: x123 + x4 + … + xn

# hw03p6: Multiadder

- multiadder(n): 返回一个可以连续调用n次的求和函数

- lambda x123: lambda x4: …: lambda xn: x123 + x4 + … + xn


lambda x12345678…: x12345678…

# hw03p6: Multiadder

- multiadder(n)：返回一个可以连续调用n次的求和函数
- lambda x123: lambda x4: …: lambda xn: x123 + x4 + … + xn

lambda x: x

lambda x12345678…: x12345678…

# hw03p6: Multiadder

- **multiadder(n)**: 返回一个可以连续调用n次的求和函数

- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

lambda x12: lambda x3: …: lambda xn: x12 + x3 + … + xn

# hw03p6: Multiadder

- multiadder(n)：返回一个可以连续调用n次的求和函数
- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

lambda x12: lambda x3: …: lambda xn: x12 + x3 + … + xn

multiadder(n - 1)

# hw03p6: Multiadder

- **multiadder(n)**: 返回一个可以连续调用n次的求和函数
- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

lambda x12: lambda x3: …: lambda xn: x12 + x3 + … + xn

multiadder(n - 1)(x1 + x2)

# hw03p6: Multiadder

- multiadder(n)：返回一个可以连续调用n次的求和函数

- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

- Base case: n == 1 => lambda x: x
- Recursion: 

# hw03p6: Multiadder

- multiadder(n): 返回一个可以连续调用n次的求和函数

- 实现 lambda x1: lambda x2: …: lambda xn: x1 + x2 + … + xn

- Base case: n == 1 => lambda x: x
- Recursion: lambda x: lambda y: multiadder(n – 1)(x + y)

# hw03p6: Multiadder

- multiadder(n)：返回一个可以连续调用n次的求和函数

- 换个思路：E.g. multiadder(5)(1)(2)
  - 总共有5个数要求和，已经运算了2个数

  - 还要加3个数，已有的数的和为3
  - multiadder的参数不够保存这些信息，需要我们定义新的函数

# hw03p6: Multiadder

- **multiadder(n)**：返回一个可以连续调用n次的求和函数

- 换个思路：（由莉莉丝提供代码）

```python
def multiadder(n):
    def helper(n, curr_sum):
        def inner(x):
            if n == 1:
                return curr_sum + x
            return helper(n - 1, curr_sum + x)
        return inner
    return helper(n, 0)
```

# hw03p8: All-Ys Has Been

Y是个什么玩意?

# hw03p8: All-Ys Has Been

Y是个什么玩意?

- 函数的不动点(fix-point)

$$\{ x \mid f(x) = x \}$$

# hw03p8: All-Ys Has Been

Y是个什么玩意?

- 函数的不动点(fix-point)

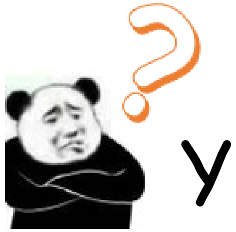
$$\{ x \mid f(x) = x \}$$

- 不动点组合子(fix-point combinator)

$$\forall f,\ Y(f) = f(Y(f))$$

对于任意函数，Y可以找到他的不动点!

# hw03p8: All-Ys Has Been



Y是个什么玩意?

- Y组合子(Y combinator)

$$Y = \lambda f.\left(\lambda x.f(x\ x)\right)\left(\lambda x.f(x\ x)\right)$$

**Haskell Brooks Curry** was an American mathematician and logician. Curry is best known for his work in combinatory logic. While the initial concept of combinatory logic was based on a single paper by Moses Schönfinkel, Curry did much of the development. Curry is a

# hw03p8: All-Ys Has Been

这是hw03p7的一种答案

- Θ组合子(Θ combinator)

$$\Theta = (\lambda x. \lambda y.\ y(x\ x\ y))(\lambda x. \lambda y. y(x\ x\ y))$$

**Alan Mathison Turing** was an English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist. Turing was highly influential in the development of theoretical computer science, providing a formalisation of the concep

# hw03p8: All-Ys Has Been

- 已知Y fib = fib (Y fib)
- fib应该填什么能得到斐波那契数列?

# hw03p8: All-Ys Has Been

- 已知Y fib = fib (Y fib)
- fib应该填什么能得到斐波那契数列?

- fib = lambda f: lambda r: 1 if r <= 1 else f(r - 1) + f(r - 2)

瞎猜的怎么就对了?

# hw03p8: All-Ys Has Been

- 已知Y fib = fib (Y fib)
- fib应该填什么能得到斐波那契数列?

- fib = lambda f: lambda r: 1 if r <= 1 else f(r - 1) + f(r - 2)

```
Y fib = fib (Y fib)
      = lambda r: 1 if r <= 1 else (Y fib)(r - 1) + (Y fib)(r - 2)
```

- Y fib = ?

# hw03p8: All-Ys Has Been

What Would Python Display?

```
>>> Y = lambda f: (lambda x: f(x(x)))(lambda x: f(x(x)))
>>> Y(fib)(3)
```

# hw03p8: All-Ys Has Been

What Would Python Display?

```
>>> Y = lambda f: (lambda x: f(x(x)))(lambda x: f(x(x)))
>>> Y(fib)(3)
```

fibonacci(3) = ___?

# hw03p8: All-Ys Has Been

What Would Python Display?

```
>>> Y = lambda f: (lambda x: f(x(x)))(lambda x: f(x(x)))
>>> Y(fib)(3)
Runtime Error: maximum recursion depth exceeded
```

# hw03p8: All-Ys Has Been

What Would Python Display?

```
>>> Y = lambda f: (lambda x: f(x(x)))(lambda x: f(x(x)))
>>> Y(fib)(3)
Runtime Error: maximum recursion depth exceeded
```

```
Y(f) = (lambda x: f(x(x)))(lambda x: f(x(x)))
     = f((lambda x: f(x(x)))(lambda x: f(x(x))))
     = f(Y f)
```

# hw03p8: All-Ys Has Been

What Would Python Display?

```
>>> Y = lambda f: (lambda x: f(x(x)))(lambda x: f(x(x)))
>>> Y(fib)(3)
Runtime Error: maximum recursion depth exceeded
```

Y(f) = (lambda x: f(x(x)))(lambda x: f(x(x)))
= f((lambda x: f(x(x)))(lambda x: f(x(x))))
= f(Y f) = f(f(Y f)) = f(f(f(Y f))) = f(f(f(f(Y f))))
= …… => Runtime Error

# hw03p8: All-~~Ys~~ Zs Has Been

- Z组合子(Z combinator)

$$Z = \lambda f.\big(\lambda x.f(\lambda z.x\ x\ z)\big)\big(\lambda x.f(\lambda z.x\ x\ z)\big)$$

$$\forall f,\ Z(f)(z) = f(Z(f))(z)$$

# hw03p8: All-~~Ys~~ Zs Has Been

- Z组合子(Z combinator)

$$Z = \lambda f.\left(\lambda x. f(\lambda z. x\ x\ z)\right)\left(\lambda x. f(\lambda z. x\ x\ z)\right)$$

∀f, Z(f)(z) = f(Z(f))(z)

通过添加变量z，巧妙地避免了Y组合子求值时的无穷递归

Z(f) = (lambda x: f(lambda z: x(x)(z)))(lambda x: f(lambda z: x(x)(z)))

= f(lambda z: (lambda x: f(lambda w: x(x)(w)))(lambda x: f(lambda w: x(x)(w)))(z))

= f(lambda z: Z(f)(z))

Zs

# hw03p8: All-~~Y~~s Has Been

- Z组合子(Z combinator)

$$Z = \lambda f.\big(\lambda x.f(\lambda z.x\ x\ z)\big)\big(\lambda x.f(\lambda z.x\ x\ z)\big)$$

$$\forall f,\ Z(f)(z) = f(Z(f))(z)$$

通过添加变量z，巧妙地避免了Y组合子求值时的无穷递归

```
Z(f) = (lambda x: f(lambda z: x(x)(z)))(lambda x: f(lambda z: x(x)(z)))
     = f(lambda z: (lambda x: f(lambda w: x(x)(w)))(lambda x: f(lambda w: x(x)(w)))(z))
     = f(lambda z: Z(f)(z))
```

这和Y(f)有什么区别?

# hw03p8: All-~~Ys~~ Zs Has Been

- Z组合子(Z combinator)

$$Z = \lambda f.\left(\lambda x. f\left(\lambda z. x\ x\ z\right)\right)\left(\lambda x. f\left(\lambda z. x\ x\ z\right)\right)$$

$$\forall f,\ Z(f)(z) = f(Z(f))(z)$$

通过添加变量z，巧妙地避免了Y组合子求值时的无穷递归

Z(f) = f(lambda z: Z(f)(z))

Z fib = lambda r: 1 if r <= 1 else (lambda z: Z(fib)(z))(r - 1) + (lambda z: Z(fib)(z))(r - 2)

# hw03p8: All-~~Ys~~ Zs Has Been

- Z组合子(Z combinator)

$$Z = \lambda f.\left(\lambda x. f(\lambda z. x\ x\ z)\right)\left(\lambda x. f(\lambda z. x\ x\ z)\right)$$

$$\forall f,\ Z(f)(z) = f(Z(f))(z)$$

通过添加变量z，巧妙地避免了Y组合子求值时的无穷递归

Z(f) = f(lambda z: Z(f)(z))

Z fib = lambda r: 1 if r <= 1 else (Z fib)(r - 1) + (Z fib)(r - 2)

# 一些学习建议

不要找助教私聊问题（聊天、心态崩了除外）

- 群里人多，并且大概率有人已经问过了
- 你的问题可能过于弱智，助教也不知道如何回答是好

- 为什么不是满分 → **亲亲，您的代码有bug哦~**
- 报错是怎么回事 → 看不懂英语的话可以买一本词典

遇到问题:

```
Traceback (most recent call last):
    return [redacted]
              ^
SyntaxError: invalid syntax
```

遇到问题：

```
Traceback (most recent call last):
    return [redacted]
           ^
SyntaxError: invalid syntax
```

←查字典：0元，还能学英语

*invalid* *在英语-中文（繁体）词典中的翻译*

# invalid

***adjective***

UK /ɪnˈvæl.ɪd/ US /ɪnˈvæl.ɪd/

**An invalid document, ticket, law, etc. is not legally or officially acceptable.**

（文件、票、法律等）無效的，不合法的，官方不承認的

*syntax* *在英语-中文（繁体）词典中的翻译*

# syntax

***noun*** [ U ] • LANGUAGE • specialized

UK /ˈsɪn.tæks/ US /ˈsɪn.tæks/

**the grammatical arrangement of words in a sentence**

句法；句子結構

遇到问题:

```
Traceback (most recent call last):
    return [redacted]
          ^
SyntaxError: invalid syntax
```

*invalid* *在英语-中文（繁体）词典中的翻译*

## invalid

*adjective*

UK /ɪnˈvæl.ɪd/ US /ɪnˈvæl.ɪd/

**An invalid document, ticket, law, etc. is not legally or officially acceptable.**

（文件、票、法律等）無效的，不合法的，官方不承認的

←查字典：0元，还能学英语

*syntax* *在英语-中文（繁体）词典中的翻译*

## syntax

*noun* [ U ] • LANGUAGE • specialized

UK /ˈsɪn.tæks/ US /ˈsɪn.tæks/

**the grammatical arrangement of words in a sentence**

句法；句子結構

↓群内提问：建议收费500元
知识付费　明码标价不坑人

# 一些学习建议

不要死磕某一道题

- 做不出就是做不出，可能是思路不对，可能是你没学会

- 一个小时都做不出的话，建议呼吸一下新鲜空气
- 再仔细读一遍题目、再看一遍ppt、录屏
- 找会做的同学/舍友问问他们的思路（如何提问.pdf）

- 放弃这道题，不要因为某一题影响整门课和其他课的学习

# Questions?

201220195 叶恒迪 2021/10/25 1:12:31

恭喜你变得更强了👍

201220098 杨林 2021/10/25 1:12:19

恭喜你变得更强了👍

201870214 沈珺妍 2021/10/25 1:05:56

恭喜你变得更强了👍

211220184 蒋知睿 2021/10/25 1:03:52

恭喜你变得更强了👍

211108100-王一安 2021/10/25 1:03:38

恭喜你变得更强了👍

215220005 翁邱一洪 2021/10/25 0:56:40

恭喜你变得更强了👍

211220177 郑恒彬🤗 2021/10/25 0:55:04

恭喜你变得更强了👍

201220195 叶恒迪 2021/10/25 0:53:55

恭喜你变的更强了👍

201870214 沈珺妍 2021/10/25 0:53:34

恭喜你变得更强了👍