CS1010S Tutorial 4

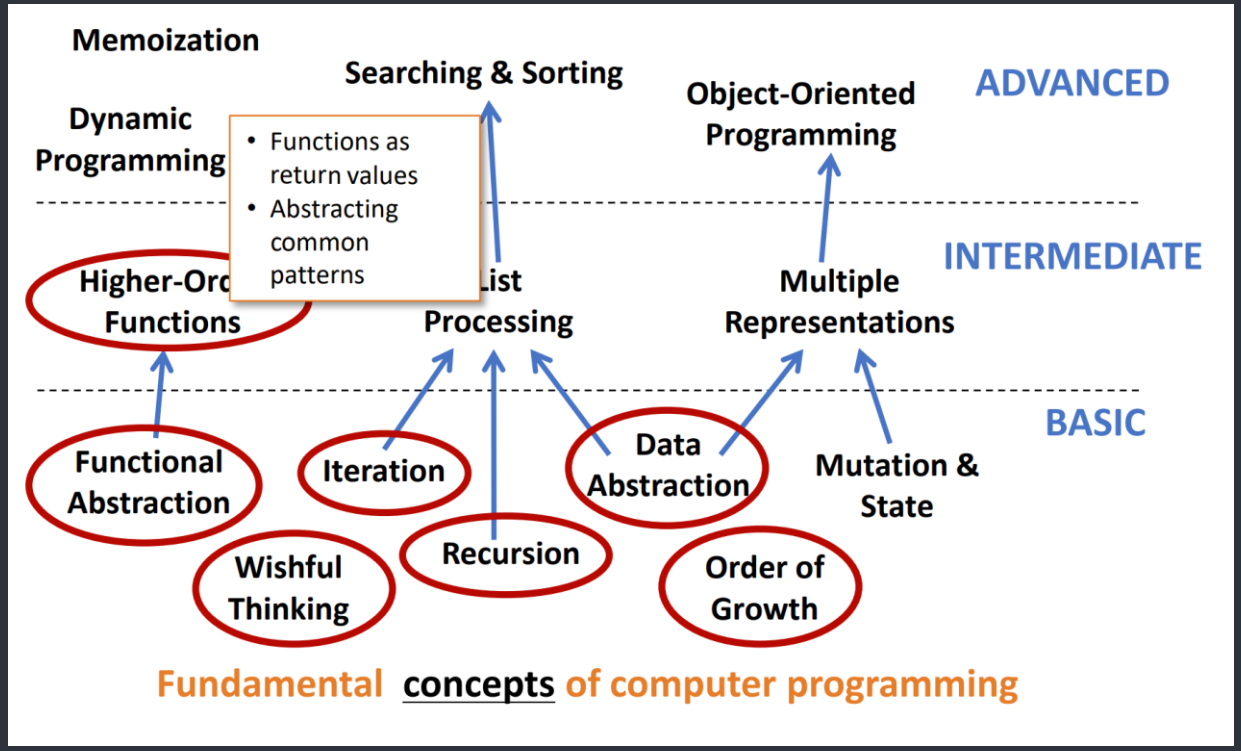
Higher-Order Functions
(and Abstract Data Types)

Lee Pak Shuang

take_note.py

take_note.py > roadmap

```
1  Lecture 5: Data Abstraction & Debugging
2  - Introduction to Data Abstraction
3  - Tuples
4  - Box and Pointer Notation
5  - Abstraction Barriers
6  - Equality (== vs is)
7  - Debugging: Errors
8
9  Highlight:
10 - Lambda Functions
11
12
13
14
15
16
17
18
```



take_note.py

take_note.py > lambda_functions

```
1  def f(<inputs>):
2      return <output>
3
4  f = lambda <inputs>: <output>
5
6  def double(x):
7      return 2 * x
8
9  double = lambda x: 2 * x
10
11 def combine(x, y):
12     return x + 2*y
13
14 combine = lambda x,y: x + 2*y
15
16
17
18
```

take_note.py

take_note.py > lambda_functions

```
1  def make_root(n):
2      def root(x):
3          return x ** (1/n)
4      return root
5
6  make_root = lambda n: lambda x: x ** (1/n)
7
8  make_root(3)
9  # <function <lambda>.<locals>.<lambda> at 0x00000203C3D5D090>
10 make_root(3)(27)
11 # 3.0
12
13
14
15
16
17
18
```

take_note.py

take_note.py > lambda_functions

```
1  (lambda x: lambda y: 2*x)(3)(4)
2  (lambda y: 6)(4)
3  # 6
4
5  def bar(f, g):
6      return lambda x: (lambda y: f(x))(g(x))
7
8  print(bar(lambda x: x+1, lambda x: x*2)(5))
9  bar(lambda x: x+1, lambda x: x*2)           (5)
10 bar(add_1, times_2)                         (5) # rename lambdas
11 (lambda x: (lambda y: add_1(x))(times_2(x)))(5) # bar return statement
12 (lambda y: (add_1)(5))(times_2(5))              # sub in 5
13 (lambda y: 6)(10)                               # evaluate the 2 lambdas
14 6                                               # y is 10 but y is not used
15
16
17
18
```

tutorial_4.py

tutorial_4.py > question_1

```python
# h = (b-a)/n, for some even integer n, and yk = f(a+kh), k = 0,1,...,n
def calc_integral(f, a, b, n):
    h = (b-a) / n

    total = 0
    for k in range(n + 1):
        term = f(a + k*h)
        if k == 0 or k == n:
            total += term
        elif k%2 == 0:
            total += 2*term
        else:
            total += 4*term

    return total * h/3

# integrate f(x) = x**3 between 0 and 1 with n = 100
print(calc_integral(lambda x : x**3, 0, 1, 100))
```

$$\frac{h}{3}[y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + y_n]$$

tutorial_4.py

tutorial_4.py > question_1

```python
# sum function as defined in lecture
def sum(term, a, next, b):
    if a > b:
        return 0
    return term(a) + sum(term, next(a), next, b)

def calc_integral(f, a, b, n):
    h = (b-a)/ n
    def term(k):
        yk = f(a + k*h)
        if k == 0 or k == n:
            return yk
        elif k%2 == 0:
            return 2 * yk
        else:
            return 4 * yk
    add1 = lambda x: x + 1
    return sum(term, 0, add1, n) * h/3
```

$$\frac{h}{3}[y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + y_n]$$

tutorial_4.py

tutorial_4.py > question_2

```python
def fold(op, f, n):
    if n == 0:
        return f(0)

    return op(f(n), fold(op, f, n-1))

# op = lambda x,y: x*y
# f(n) * (f(n-1) * (f(n-2) * (... * (f(0)))))

def g(k):
    return fold(lambda x,y: x*y, lambda x: x - (x+1)**2, k)
```

$$g(k) = \prod_{x=0}^{k} (x - (x+1)^2)$$

tutorial_4.py

tutorial_4.py > question_3a

$$a_1 = a,\ a_n \leq b$$

$$accumulate(\oplus, base, f, a, next, b):\ (f(a_1) \oplus (f(a_2) \oplus (... \oplus (f(a_n) \oplus base)...)))$$

```
1
2
3
4
5
6
7  def accumulate(combiner, base, term, a, next, b):
8      if a > b:
9          return base
10     return combiner(term(a), accumulate(combiner, base, term, next(a), \
11             next, b))
12
13 # define sum in terms of accumulate
14 def sum(term, a, next, b):
15     return accumulate(lambda x,y: x+y, 0, term, a, next, b)
16
17
18
```

tutorial_4.py

tutorial_4.py > question_3b

$$a_1 = a,\ a_n \leq b$$

$$accumulate(\oplus, base, f, a, next, b):\ (f(a_1) \oplus (f(a_2) \oplus (... \oplus (f(a_n) \oplus base)...)))$$

```
1
2
3
4
5  def accumulate_iter(combiner, base, term, a, next, b):
6      # generate all the terms and put them in reverse order
7      terms = ()
8      while a <= b :
9          terms = (term(a),) + terms
10         a = next(a)
11     # terms = (f(an), ..., f(a1))
12
13     # combine the terms
14     result = base
15     for term in terms:
16         result = combiner(term, result)
17
18     return result
```

tutorial_4.py

tutorial_4.py > question_4ai_4aii

```
1  def make_point(x,y): # setter
2      return (x, y)
3
4  def x_point(p): # getter
5       return p[0]
6
7  def y_point(p): # getter
8       return p[1]
9
10 def make_segment(p1,p2): # setter
11     return (p1, p2)
12
13 def start_segment(seg): # getter
14      return seg[0]
15
16 def end_segment(seg): # getter
17      return seg[1]
18
```

tutorial_4.py

tutorial_4.py > question_4aiii

```
1  def midpoint_segment(segment):
2      start_point = start_segment(segment)
3      end_point = end_segment(segment)
4
5      mid_x = 0.5 * (x_point(start_point) + x_point(end_point))
6      mid_y = 0.5 * (y_point(start_point) + y_point(end_point))
7
8      return make_point(mid_x, mid_y)
9
10
11
12
13
14
15
16
17
18
```

tutorial_4.py

tutorial_4.py > question_4b

```
1  # Implement a representation for a rectangle in a 2D plane
2  rect = lambda l_seg, w_seg: (l_seg, w_seg) # setter
3
4  rect_l_seg = lambda rect: rect[0] # getter
5  rect_w_seg = lambda rect: rect[1] # getter
6  rect_l = lambda rect: seg_length(rect_l_seg(rect))
7  rect_w = lambda rect: seg_length(rect_w_seg(rect))
8
9  import math
10 def seg_length(seg):
11     start_pt = start_segment(seg)
12     end_pt = end_segment(seg)
13     x_delta = x_point(start_pt) - x_point(end_pt)
14     y_delta = y_point(start_pt) - y_point(end_pt)
15     return math.sqrt(x_delta*x_delta + y_delta*y_delta)
16
17 perimeter = lambda rect: 2*rect_l(rect) + 2*rect_w(rect)
18 area = lambda rect: rect_l(rect) * rect_w(rect)
```

conclusion.txt

conclusion.txt

```
1  Lecture 6: Working with Sequences
2
3  Mission 5: due TOMORROW
4
5  Coursemology: Aim for level 26-33
6
7  This week: SQ5.X, M6, M7, SQ7.1, M8
8
9  Recess week: SQ8.X, Midterm PYPs
10
11 Consultations: DM me to schedule
12
13 Week 7 tutorial slot: Midterms Prep
14
15 Plagiarism: pls don't
16
17
18
```