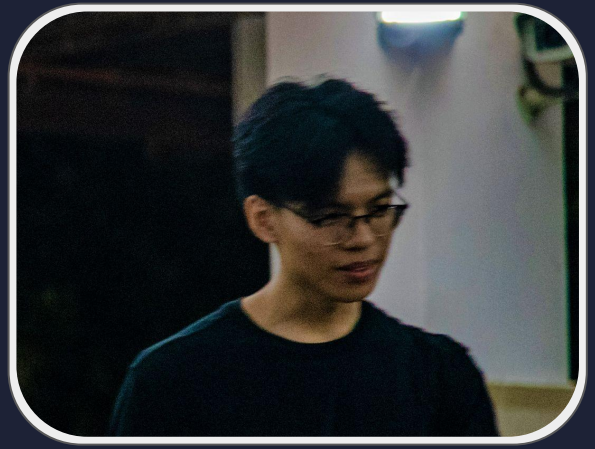
BY: Zhu_Ming

<CS1010S>

Tutorial 1

Introduction to Python

Self_Introduction.Tut01



HOR ZHU MING

Year 3 PHYSICS Undergraduate
Minor in Computer Science

No programming background before CS1010S!

Telegram_Helpline: **@Zhuming05**
Email: **e0735464@u.nus.edu**

Lecture Recap

1. Data type
   integer, float, string, boolean
   type conversion
2. Variable assignment
   ```
   a = 1
   class = "1010s"
   ```
3. Arithmetic operators
   `+  -  *  /  **  //  %`
4. Comparison operators
   `>  <  >=  <=  ==  !=`
5. Logical operators
   `and  not  or`
6. Conditional Statements
   `if  else  elif`
7. Indexing
   ```
   class[0] = "1"
   class[-1] = "s"
   ```
8. Slicing
   ```
   class[1:4] = "010s"
   ```
9. Functions
   ```
   def add_one(x):
       return x + 1
   ```
10. Functional Abstraction
11. Divide and Conquer

Lecture_Recap.tut01

1. Float (Decimal Number)

```
>>> 1 / 3 # 0.3333333333333333
```

***By default, Python can handle up to 16dp*

2. Booleans

```
True, False # capital letter
```

3. Type Conversion

```
my_int = 1
my_string = str(my_int) # “1”

type(my_int) # <class ‘int’>
type(my_string) # <class ‘str’>

>>> a = 1 + 1 # 2
>>> b = “1” + “1” # “11”
>>> c = “1” + “   1” # “1   1” **Spacing matters in string!!
```

4. **Indexing & Slicing** (0-based indexing)

my_string[start : stop : step]
**included start but excluded stop*

```
word = “abcdefg”
p = word[0] # “a”
x = word[-1] # “g”
y = word[1:3] # “bc”
z = word[1:5:-1] # “edcb”
```

a	b	c	d	e	f	g
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

5. = vs ==

```
a = 1  # variable assignment
a == 1 # comparison
```

6. Truthy / Falsy value

Falsy: 0, “”, (), []
Truthy: any non-zero number, non-empty container

```
bool(0) # False
bool(1) # True
bool(“is True”) # True
bool([]) # False

const = 5
if conts:
    print(“const is truthy”)
```

7. Function

```
def <name>(<parameters>):
    <body>
    return <statements>
```

Define function add_one which take one argument,x
return x + 1 when argument is positive, else return x

```
def add_one(x):
    if x > 0:
        return x + 1
    else:
        return x
```

7. Condition

```
if <cond_1>:
     <statements_1>
elif <cond_2>:
     <statements_2>
elif <cond_2>:
     <statements_2>
.
.
. n-times
else:
     <statements_n>
```

8. Block

```
if <cond_1>:
     <statements_1>
if <cond_2>:
     <statements_2>
if <cond_3>:
     <statements_3>
```

```
if <cond_1>:
     <statements_1>
elif <cond_2>:
     <statements_2>
else:
     <statements_3>
```

BY: Zhu_Ming

Any Questions?

Question_1.tut01

```python
def square(x):
    return x ** 2

print(square(2)) # 4
print(square(4)) # 16
print(square(square(square(2)))) # 256


def f(x):
    return x * x

print(f(4)) # 16
```

python_tip_1.tut01

IMPORTANT

Python interpret expression from inner () to the outer ()

Is similar to Algebra!

Question_1.tut01

```
def try_f(f):
    return f(3)

print(try_f(f)) # 9
print(try_f(f) == try_f(square)) # True
print(f(3) == square(3)) # True
print(f == square) # False



>>> f
<function f at 0x000001F710EF8820>

>>> square
<function square at 0x000001F710EF8EE0>
```

python_tip_2.tut01

- Integer comparison
  - Compares their values
- Function comparison
  - Compares their memory address

BONUS

- String comparison
  - Compares their ASCII value

‘0’<‘9’<‘A’<‘Z’<‘a’<‘z’

Question_2.tut01

Using if-else, define a function odd(x) that returns True when its integer argument is an odd number and False otherwise.

Version 1

```
def odd(x):
    if x % 2 == 1:
        return True
    else:
        return False
```

Question_2.tut01

Use **Truthy-Falsy Value**

Compact Version

```
def odd(x):
    if x % 2:
        return True
    else:
        return False
```

Question_3.tut01

Now, **without** using if-else, define the function new_odd(x) that does the same.

Version 1

```
def new_odd(x):
    return x % 2 != 0
```

Question_3.tut01

Use bool function

Compact Version

```
def odd(x):
    return bool(x%2)
```

Question_4.tut01

Define a function that will return the number of digits in an integer. You can safely assume that the integers are non-negative and will not begin with the number 0 other than the integer 0 itself.

Method 1: Recursion (Eazy)

```
def number_of_digits(i):
    if abs(i) < 10:
        return 1
    else:
        return 1 + number_of_digits(
                                i // 10)
```

Question_4.tut01

Method 2: Iteration (Challenging)

```
def number_of_digits(i):
    if x == 0:
        return 1
    ans = 0
    while x > 0:
        x //= 10
        ans += 1
    return ans
```

Method 3:

Use `len()` function

```
def number_of_digits(i):
    return len(str(i))
```

Question_5.tut01

Define a function that takes **THREE numbers as arguments** and returns the SUM of the SQUARE of the TWO LARGER NUMBER.

Solution 1

```
def bigger_sum(a, b, c):
    if a <= b and a <= c: # smallest = a
        return b * b + c * c
    elif b <= a and b <= c: # smallest = b
        return a * a + c * c
    else: # smallest = c
        return b ** 2 + a ** 2
```

Question_5.tut01

But this is BAD example :(

Why??

Question_5.tut01

Use **Functional Abstraction & Divide and Conquer!!**

Good Solution

```
def square(x):
    return x ** 2

def sum_of_square(x, y):
    return square(x) + square(y)

def bigger_sum(a, b, c):
    if a <= b and a <= c:
        return sum_of_square(b, c)
    elif b <= a and b <= c:
        return sum_of_square(a, c)
    else:
        return sum_of_square(b, c)
```

Question_5.tut01

Alternative Solution

Use min() function

```
>>> min(1, 2, 3) # 1
```

```
def bigger_sum(a, b, c):
    min_i = min(a, b, c)
    return square(a)
        + square(b)
        + square(c)
        - square(min_i)
```

*** We use function square() here. Always use Functional Abstraction. Very important concept!!*

Question_6.tut01

Write a function **is_leap_year** that takes **ONE integer parameter** and decides whether it corresponds to a leap year.

Year is Leap Year if:-

1. Year is multiples of 400
2. Year is multiples of 4, but NOT multiples of 100
3. Otherwise, Not a Leap Year

Solution

```
def is_leap_year(year):
    if year % 400 == 0:
        return True
    elif year % 4 == 0 and year % 100 != 0:
        return True
    else:
        return False
```

Compact Solution

```
def is_leap_year(year):
    if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
        return True
    else:
        return False
```

MORE COMPACT Solution

```
def is_leap_year(year):
    return (year % 400 == 0) or (year % 4 == 0 and \ year % 100 != 0)
```

Extra_Question.tut01

EXTRA Practices

EXTRA_Question.tut01

(EXTRA)

Global & Local Scope

QUESTION 1

```
x = 1

def foo(x):
    return x + 1

print(foo(2))
print(foo(x))
print(x)
```

EXTRA_Question.tut01

OUTPUT:

```
3
2
1
```

EXTRA_Question.tut01

(EXTRA)

Global & Local Scope

Question 2

```
y = 1

def bar(x):
    x = 2
    return x + 2

print(bar(5))
print(bar(y))
print(y)
print(x)
```

EXTRA_Question.tut01

OUTPUT:

```
4
4
1
NameError (Python: harr, what is x??)
```

EXTRA_Question.tut01

(EXTRA)

Global & Local Scope

Question 3

```
z = 1

def meh(y):
    x = 2
    return y + 2

print(meh(5))
print(meh(y))
print(z)
```

EXTRA_Question.tut01

OUTPUT:

7
Error # What is y??

***take note that print(z) is never executed because Python stop running when there is an error*

EXTRA_Question.tut01

(EXTRA)

print VS return

```
1   def return_only(x):
2       return x
3
4   def print_only(x):
5       print(x)
6       # return None
7
8   test_1 = return_only(4)
9   test_2 = print_only(5)
10
11  print(test_1)
12  print(test_2)
13  print(print_only(5))
```

EXTRA_Question.tut01

OUTPUT:

```
5 # due to line9
4 # due to line11
None # due to line12
5 # due to line12
None # due to line13
```

BY: Zhu_Ming

Any Questions?

Thank You!!

The End

See you next lesson