INTRODUCTION TO PYTHON

Grading Policy

General performance	10%	
In-class practice test	20%	
1st midterm exam	20%	(乙) 3/21 (甲) 3/22
2nd midterm exam	20%	(乙) 5/02 (甲) 5/03
Final exam	30%	(乙) 6/13 (甲) 6/14

Arithmetic in Python

- Basic numeric operators include
  - `+` addition
  - `-` subtraction
  - `*` multiplication
  - `/` division
  - `**` exponentiation
  - `%` modulus: gives the remainder of a division

Data Types and Operators

- There are really two sets of numeric operators
  - One for integers (int)
  - One for floating-point numbers (float)
- In most cases, the following rules apply
  - If *at least one* of the operands is a float, the result is a float
  - If *both* of the operands are ints, the result is an int
- One exception: Division

Arithmetic in Python

- The operators follows the PEMDAS (Parentheses, Exponents, Multiplication/Division, Addition/Subtraction) order of operations
- Exceptions:
  - Multiplications and divisions are evaluated from left to right
  - Additions and subtractions are evaluated from left to right
- Recall PEMDAS
  - $2/2 + 1 * 3$

    4
  - $2/(2 + 1) * 3$

    2

Numeric Data Types

- Different kinds of values are sorted and manipulated differently
- Python data types include
  - Integers
    - Example: 389
  - Float-point numbers
    - Example: 3.14159

Two Types of Division

- The / operator always produces a float result
  - Examples
    - 5/3
      1.6666666666666667
    - 6/3
      2.0

Two Types of Division

- There is a separate // operator for integer division
  - 6//3
    2
- Integer division *discards* any fractional part of the result
  - 11//5
    2
  - 5//3
    1
- Note that it does not round; only the "whole part" of the division is returned (*floor* function or *truncation*)

Another Data Type

- A string is a sequence of characters/symbols
- Surrounded by single or double quotes
  - Examples
    - ‘hello world.’
    - “FCU”
    - ‘3.14159’

Variables

- Variables allow us to store a value for later use
  - temp = 73
  - temp = temp – 5
  - (temp – 32) * 5 / 9
- Updating a variable requires assignment to a new value
  - temp = 86

Expressions

- Expressions produce a value
  - Python evaluates them to obtain their value
- They include
  - Literals
    - 3.14159
    - ‘university’
  - Variables
    - temp
  - Combinations of literals, variables, and operators
    - (temp – 32) * 5 / 9

Evaluating Expressions with Variables

- When an expression includes variables, they are first replaced with their current value
- Example to show how Python would evaluate this

```
(temp – 32) * 5 / 9
(68 – 32) * 5 / 9
36 * 5 / 9
180 / 9
20.0
```

Statements

- A statement is a command that carries out an action
- A program is a sequence of statements

```
fifty = 3
ten = 6
five = 2
one = 8
amount = 50 * fifty + 10 * ten + 5 * five + one
print('You have', amount, 'dollars.')
```

Assignment Statements

- Assignment statements store a value in a variable
  - temp = 73
- General syntax: **variable = expression**
- 「=」 is known as assignment operator
- Steps
  ① Evaluate the expression on the RHS of the 「=」
  ② Assign the resulting value to the variable on the LHS of the 「=」
- Example

```
fifty = 3
fifty_value = 50 * fifty
fifty_value = 50 * 3
fifty_value = 150
```

Assignment Statements

- We can change the value of a variable by assigning it a new value
- Example

num1 = 100 num2 = 120	num1 100 num2 120
num1 = 50	num1 num2
num1 = num2 * 2	num1 num2
num2 = 60	num1 num2

Assignment Statements

- We can change the value of a variable by assigning it a new value
- Example

num1 = 100 num2 = 120	num1 100 num2 120
num1 = 50	num1 50 num2 120
num1 = num2 * 2 120 * 2 240	num1 240 num2 120
num2 = 60	num1 240 num2 60

Assignment Statements

- A variable can appear on both sides of the assignment operator
- Example

sum = 13 val = 30	sum 13 val 30
sum = sum + val	sum val
val = val * 2	sum val

Assignment Statements

- A variable can appear on both sides of the assignment operator
- Example

sum =	13				
val =	30	sum	13	val	30
sum =	sum + val 13 + 30 43	sum	43	val	30
val =	val * 2 30 * 2 60	sum	43	val	60

Creating a Reusable Program

- Put the statements in a Jupyter Notebook

```
# A program to compute the value of some coins

fifty = 3
ten = 6
five = 2
one = 8
amount = 50 * fifty + 10 * ten + 5 * five + one
print('You have', amount, 'dollars.')
```

- Rename the file to *coins*

Print Statements

- Print statements display one or more values on the screen
- Basic syntax
  ① print($expr$)
  ② print($expr_1$, $expr_2$, …, $expr_n$)
- Steps
  ① The individual expression(s) are evaluated
  ② The resulting values are displayed on the same line, *separated by spaces*
- To print a blank line, use **print()**

Print Statements

- Example 1

```
print('The results are:', 15 + 5, 15 - 5)
                  ↓          ↓      ↓
          'The results are:'  20     10
```

  - Output: *The results are: 20 10*
  - Note that the quotes around the string literal are **not** printed

Print Statements

- Example 2

  cents = 89

  print(‘You have’, cents, ‘cents.’)

  ↓ ↓ ↓

  ‘You have’ 89 cents.

  - Output: *You have 89 cents.*

Variables and Data Types

- The type function gives us the type of an expression
  - type(‘hello world.’)
    <class ‘str’>
  - type(5 / 2)
    <class ‘float’>
- Variables in Python do **not** have a fixed type
  - temp = 86.0
  - type(temp)
    <class ‘float’>
  - temp = 68
  - type(temp)
    <class ‘int’>

How a Program Flows

- Flow of control = the order in which statements are executed
- By default, a program's statements are executed sequentially, from top to bottom

Example program	*Variables in memory*
total = 0	total num1
num1 = 5	num2
num2 = 10	
total = num1 + num2	

How a Program Flows

- Flow of control = the order in which statements are executed
- By default, a program's statements are executed sequentially, from top to bottom

Example program	*Variables in memory*
total = 0 num1 = 5 num2 = 10 total = num1 + num2 5 + 10 15	total 15 num1 5 num2 10

What is the Output of the Following Program?

```
x = 15
name = ‘FCU’
x = x // 2
print(‘name ’, x, type(x))
```

```
x = x // 2
  = 15 // 2
  = 7
```

‘name ’ 7 type(7)

<class ‘int’>

A. FCU 7 <class ‘int’>
B. FCU 7.5 <class ‘float’>
C. name 8 <class ‘int’>
D. name 7 <class ‘int’>
E. name 7.5 <class ‘float’>

What is the Output of the Following Program?

```
x = 15
name = 'FCU'
x = x // 2
print('name ', x, type(x))
```

```
x = x // 2
  = 15 // 2
  = 7
```

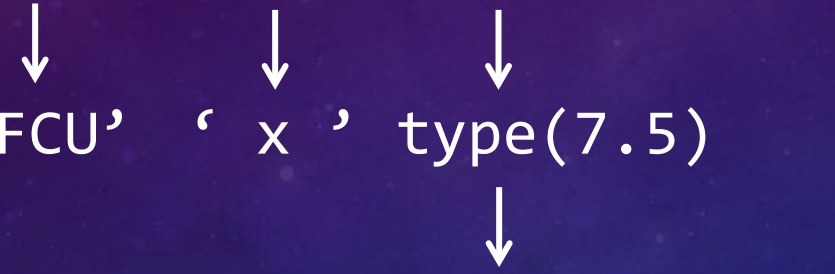
'name ' 7 type(7)

<class 'int'>

A. FCU 7 <class 'int'>
B. FCU 7.5 <class 'float'>
C. name 8 <class 'int'>
D. name 7 <class 'int'>
E. name 7.5 <class 'float'>

What about This Program?

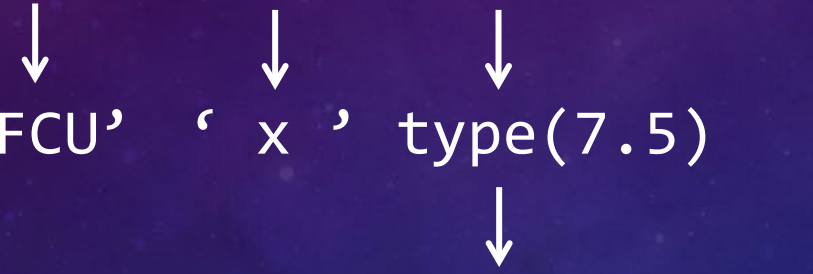
```
x = 15
name = ‘FCU’
x = 7.5
print(name, ‘ x ’, type(x))
```



A. name x <class ‘float’>
B. FCU 7.5 <class ‘float’>
C. FCU x <class ‘float’>
D. FCU 15 <class ‘int’>
E. name 7.5 <class ‘str’>

What about This Program?

```
x = 15
name = ‘FCU’
x = 7.5
print(name, ‘ x ’, type(x))
```



A. name x <class ‘float’>
B. FCU 7.5 <class ‘float’>
C. FCU x <class ‘float’>
D. FCU 15 <class ‘int’>
E. name 7.5 <class ‘str’>

What Are the Values of the Variables?

```
x = 5
y = 6
x = y + 3
z = x + y
x = x + 2
```

x	y	z
5		
5	6	
9	6	
9	6	15
11	6	15

Change the value of x does not change the value of z! (why?)

	x	y	z
A.	11	6	15
B.	11	6	11
C.	11	6	17
D.	7	6	11

E. None of the above, because the code has an error

What Are the Values of the Variables?

```
x = 5
y = 6
x = y + 3
z = x + y
x = x + 2
```

x	y	z
5		
5	6	
9	6	
9	6	15
11	6	15

Change the value of x does not change the value of z! (why?)

	x	y	z
A.	11	6	15
B.	11	6	11
C.	11	6	17
D.	7	6	11

E. None of the above, because the code has an error