LECTURE 3

COMMENTS, VARIABLES, INPUT

MCS 260 Fall 2021

Emily Dumas

COURSE BULLETINS

- Ask for help ASAP if you don't have Python and Visual Studio Code working
- Worksheet 1 solutions posted
- Homework 1 available in Gradescope; due at 10am Tuesday August 31

COMMENTS

In a line of Python code, anything appearing after a # character is ignored by the interpreter.

```
print("Hello world!")    # TODO: Choose a new greeting
```

The ignored text is a **comment**. Comments are used to explanatory text for use by humans.

A comment can take up an entire line, and this is often used to add a header at the top of a script.

```
# Hello MCS 260 script by Emily Dumas
# Written on 2021-08-26
print("Hello world!")
```

VARIABLES AND ASSIGNMENTS

Variables provide a named place to store values. The value stored in a variable can be changed later.

To set the value of a variable we use an **assignment statement**. The basic syntax is

$$\texttt{name} = \texttt{value}$$

Example:

```
>>> side_length = 5
>>> side_length
5
>>> side_length**2
25
>>> side_length = 6
```

```
>>> side_length**2
36
```

Note: Variable names don't have quotes around them.

```
"a" = 50     # FAILS: LHS is a value, not a variable name
a = 50       # Works
a = thing    # FAILS: thing is an unknown variable name
a = "thing"  # Works
b = "uic"    # Works, b is now "uic"
b = a        # Works, b is now "thing"

print(b) # The current value of variable b
         # appears on the screen
```

The right hand side of an assignment can be an **expression** combining variables, literals, function calls, and operators. These are evaluated before assignment.

```
>>> old_semester_tuition = 4763
>>> semester_tuition = old_semester_tuition * (1 + 11.1/100)
>>> semester_tuition
5291.693
```

Spaces around = are optional.

Variable name prohibitions:

- Must not start with a number
- Must not contain spaces
- Must not be a Python keyword (`if`, `while`, ...)

The Python 3.9 keywords are:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Variable name recommendations:

- Use only `A-Z`, `a-z`, 0-9, and _ (underscore)
- Use _ as a word separator

```
class_avg = 93.8     # Works
260avg = 93.8        # FAILS: starts with a number
secret code = 12345 # FAILS: spaces prohibited
secret_code = 12345 # Works
SecretCode = 12345  # Works, atypical style
测试成绩 = "great"     # Works, not recommended
```

(The exact rules for which characters can appear in variable names are rather complicated.)

TYPES

Every object in Python (whether a variable or a literal) has a **type**. You can determine the type using the built-in function `type()`:

str means string, a sequence of characters

```
>>> type("Hello world!")
<class 'str'>
```

int means integer

```
>>> type(77)
<class 'int'>
```

float means floating-point number

```
>>> type(0.1)
<class 'float'>
```

DYNAMIC TYPING

In Python, you are free to change the type of a variable at any time.

Many languages don't allow this!

```
x = 5           # x is an int
x = 3.14159     # now it's a float
x = "umbrella" # now it's a string
```

MORE ABOUT PRINTING

The `print()` function can accept any number of values, of any types, in a comma-separated list.

The basic syntax is
`print(val1, val2, val3, ...)`.

```
>>> print("The decimal value of binary 1001 is",0b1001)
The decimal value of binary 1001 is 9
>>> print("The sum of",99,"and",0b10,"is",99+0b10)
The sum of 99 and 2 is 101
>>> print(1,1.0,1+0j)
1 1.0 (1+0j)
>>>
```

In the output, values are separated by spaces.

INPUT

The `input()` function waits for the user to type a line of text in the terminal, optionally showing a prompt.

Then, the place where `input()` was called gets replaced with the string the user entered.

```
>>> s = input("Enter some text: ")
Enter some text: organizing heliotrope <--- keyboard input
>>> print("You entered:",s)
Your entered: organizing heliotrope
>>> input()
programming exercises  <--- keyboard input
'programming exercises'
>>>
```

GREETING THE USER

Let's write a program that will ask the user for their name, and then display a greeting.

ARITHMETIC ON INPUT?

We can't do arithmetic on input directly, because the input is always a string.

```
>>> 5 + input("Enter a number: ")
Enter a number: 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Instead we need to convert input to a numeric type, using `int()`, `float()` or `complex()`.

```
>>> 5 + int(input("Enter a number: "))
Input: 10
15
```

The conversion functions `int()`, `float()`, `complex()` can convert from strings to numeric types, and between numeric types, e.g.

```
>>> float(42)
42.0
>>> int(12.9)
12
```

Supported conversions:

input type →	**str**	**int**	**float**	**complex**
int()	✓	✓	✓ integer part	✗
float()	✓	✓	✓	✗
complex()	✓ picky	✓	✓	✓

RECTANGLE AREA AND PERIMETER

Let's write a script to compute the area and perimeter of a rectangle.

It will ask the user for the dimensions using `input()` and then print the results.

REFERENCES

- In *Downey*: variables and assignment statements are discussed in Chapter 2, conversion functions (int etc.) in Section 3.1, and keyboard input is covered in Section 5.11.

ACKNOWLEDGEMENTS

- Some of today's lecture was based on teaching materials developed for MCS 260 by Jan Verschelde.

REVISION HISTORY

- 2020-08-26 Initial publication.