LECTURE 3

VARIABLES, ASSIGNMENTS, INPUT

MCS 260 Fall 2020

Emily Dumas

REMINDERS

- Complete worksheet 1 this week
  - Not collected; solution recently posted
- Quiz 1 due Monday 6pm Central
  - Contact staff about install problems
  - Can request to be excused once per calendar month by writing to TA (see syllabus)
  - Upload separate images or single PDF with BIG screenshot

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 should be added where they will make code easier to understand (for others, or for you in the future). They can also be reminders about known problems or future plans, if these are not recorded systematically elsewhere.

VARIABLES AND ASSIGNMENTS

Variables allow you to give names to values, and to later change the value associated with a name. We do so with **assignment statements**. 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
```

A common mistake for beginners is to put quotation marks around variable names, or to omit them when a string is needed.

```
"foo" = 50     # FAILS: can't assign to string
foo = 50       # Works
foo = thing    # FAILS: thing is seen as variable name
foo = "thing"  # Works
bar = "foo"    # Works, bar is now "foo"
bar = foo      # Works, bar is now "thing"

print(Hello world) # FAILS: Hello and world are unknown names
                   # and space between var names not allowed
```

HOW TO THINK ABOUT ASSIGNMENT

The code

```
x = 15
```

asks Python to remember three things:

- the value 15
- the name x (i.e. create a new name if needed)
- that x currently refers to 15

A diagram is often used to summarize this situation:

$$\boxed{x} \longrightarrow \boxed{15}$$

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 3.8 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 are complicated and refer to a number of other documents and standards. Ultimately, there are about 120,000 characters allowed.)

INPUT

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

It then **returns** the text that was read, meaning that the code behaves as though that instance of `input()` has been replaced by the string the user typed.

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

GREETING THE USER

The script greeting.py will greet you by name.

```
# Greet the user by name
# MCS 260 Fall 2020 Lecture 3 - Emily Dumas
name = input("Enter your name: ")
print("Nice to meet you,", name)
```

Usage:

```
$ python greeting.py
Enter your name: Emily Dumas
Nice to meet you, Emily Dumas
$
```

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	✓	✓	✓

What "picky" means: `complex()` requires the format $x+y\texttt{j}$ or $(x+y\texttt{j})$.

```
complex("1+2j")   # Works
complex("(1+2j)") # Works
complex("2j+1")  # Fails
```

Warning: Conversion from int to float or int to complex may be destructive; ints are exact, but float and complex may replace input with an approximation.

```
>>> float(9_007_199_254_740_992)
9007199254740992.0
>>> float(9_007_199_254_740_993)
9007199254740992.0
```

SUM AND PRODUCT SCRIPT

Here is a script sumprod.py that reads two floats from the user and prints their sum and product.

```
# Read two floats and print their sum and product
# MCS 260 Fall 2020 Lecture 3 - Emily Dumas
x = float(input("First number: "))
y = float(input("Second number: "))
print("Sum:    ",x,"+",y,"=",x+y)
print("Product:",x,"*",y,"=",x*y)
```

Usage:

```
$ python sumprod.py
First number: 1.2
Second number: -3.45
Sum:     1.2 + -3.45 = -2.25
Product: 1.2 * -3.45 = -4.14
```

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

- 2020-08-28 Code formatting correction
- 2020-08-27 Initial publication