Objects, Variables, Data Types, Expressions, and Assignments

zyBooks
"Programming in Python 3"
chapter 2

Data Types

- Computer programs manipulate quantities (or "values") to produce useful results
- Quantities have a *type*
  - Numeric: **3**, **-7**, **8.5**, and so forth
  - textstring: **"abcd"**, **"Hello, world"**, **"3.14"**
    - Strings are marked by single or double *quotes*
  - More complicated types, with multiple parts
    - Lists, Tuples, Dictionaries, Sets
    - Programmers can defined others as needed

Data Objects

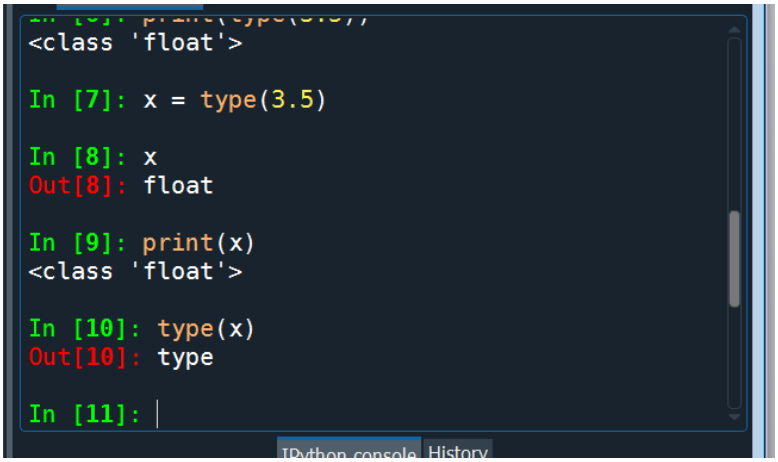
- Python refers to quantities that have known types as *objects*
  - 8.5 is a *floating-point* object (a number)
  - -7 is an *integer* object (a number)
  - "Hello world" is a *string* object
  - "M" is a one-character-long string object
  - "3.14" is a string object that looks like a number
    - But it isn't a number, it's string
- Objects in Python have "identities" that are reported by the "id( )" function
  - Technical: the identity is related to the object's location in computer RAM

Type Examples

- What type is:
  - 2
  - integer
- What type is:
  - 2.0
  - floating-point
- What type is:
  - "2"
  - string
- Mathematics – what type is:
  - 2 + 3j
  - complex number
- Mathematics – what type is:
  - "2 + 3j"
  - another string

Verify: Python Supports Type Checking

- The "type" builtin function



Variables and Constants

- Objects like 8.5 and "M" don't change their value – they are constants, or **literals**
- Some objects need to change their value as part of a program – they are **variables**
  - Variables act as "containers" for other objects
- Variables have *names* that follow rules
  - Names *start* with a letter: 'a'-'z' or 'A'-'Z'
  - Names are made up of
    - letters – 'a'-'z', 'A'-'Z',
    - digits – '0'-'9',
    - and underscore – '_'

Variable Names

- Convenient variable names are short and easy to type
  - …but it's difficult to remember what they contain
- Helpful variable names describe the values they contain
  - …but too much description is hard to type
- "Good" variable names are descriptive but easy to type
- "Best practice" says that good variable names are composed of words separated by '_'
  - '_' not always used

Examples of Variable Names

- What kind of variable names are:
  i
  a
  - *Very* temporary variables – common for brief use, but not descriptive or "good"
- How about:
  average_speed
  - Pretty good, not too hard to type
- This?
  x,y
  - x,y – *two* variables, not one!

Special Names

- Some names have predetermined meanings:
  - `print, len, list, int, str` – for example
  - These will cause confusion if they are used as variable names
- Some names are *reserved*, and cannot be changed:

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

Basic Assignments

- Assignment gives a variable a value
  i.e., it "puts an object into a container"
- Assignments use the "equals" operator
  - **=**
  - This is **different** from how the equals sign is used in mathematics
- A variable must be on the left side of the assignment
- Examples:
  - `average_speed = 65.5`
  - `myname = 'Abraham Lincoln'`

Expressions

- Expressions assemble values from other values
  - The assembled values will have a *type*
- Expressions are composed of objects and *operators*
  - Trivially small "expressions" don't need any operators
- Expression objects can be literals, variables

Some Examples of expressions

Expression	Result Type	*kind*
`3 + 4`	integer	*constant*
`(a - 8.5)/2.0`	floating-point	*variable (if a is a variable containing a floating-point object; otherwise, an error)*
`z = 'wacka '` `z * 3`	string	*The first line is an **assignment**; the second line is an **expression***
`z`	string (z as above)	*trivial – no operators*
`"3.14" – 3.0`	error!	*Can't combine numbers and string this way*

Assignments

- Expressions can be used anywhere a value is needed
- For convenience, expressions are often assigned to variables
  - Variable holds *current* value of the expression
- Examples:

```
first_name = 'Margaret'
last_name = 'Hamilton'
computer_scientist = first_name + ' ' + last_name
first_name = 'Englebert'

print(computer_scientist)  # still shows "Margaret Hamilton"
```

Updating Variables

- What does this do?

```
count = 2
print(count, end=', ')
count = count + 2
print(count, end=', ')
count = count + 2
print(count, end=', ')
count = count + 2
print(count, end=', ')
print('Time to stop and cogitate!')
```

Some Special Assignments

- Combine an operator with an assignment:
  - `x += 2` `# x's old value is incremented by 2`
  - `echo = 'Hello '` `# simple assignment`
    `echo += 'world'` `# assignment with joining`
  - `echo *= (4 + n)` `# at least 4 echoes?`
- Contents of variable on left-hand-side are *modified* by the operation and value on the right-hand-side
  - Right-hand-side can be literal or expression

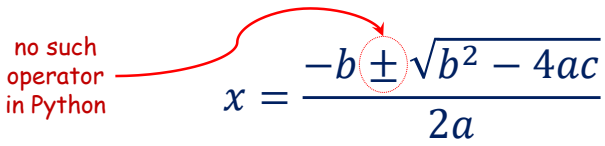
Comments – the Hashtag Symbol

- Sometimes a code writer wants to leave a note, usually explaining what some code is doing
- Lines that start with a "**#**" are *comments*
  - Extends to the end of the line
  - Can follow "real" Python statement
  - Multiline comments must have a "**#**" on each line
- Examples on previous slide

Long Expressions – the Continuation Character "\"

- Long expressions are difficult to read, and harder to debug
- Spread expression over multiple lines:
  - End first line with a backslash ("\")
  - Repeat on as many lines as needed
  - Last line ends with **no** backslash
- Backslash also used in strings to "escape" certain characters
  - "\n" – newline character
- Don't confuse backslash "\" with division "/"
  - Unrelated

Example: the Quadratic Formula



- Python versions:

```
xplus = (-b + ( b**2 – 4*a*c )**0.5) \
     / (2*a)

xminus = (-b - ( b**2 – 4*a*c \
     )**0.5) / (2*a)
```

backslashes

Continuation Details

- Backslash must be the last character on the line
  - No following comment!
- Indentation of continued lines not important
  - Usually indented to make visual recognition easier
- Preferred: continue an expression at some natural breaking point
  - Previous "xplus" break is better than "xminus" break
- Can't continue comments
  - Just make multiline comments

Some Continuations are Implied

- BMI calculation:
  - `bmi = kilos / meters**2` `# fine expression`
  - `bmi = kilos /` `# illegal syntax`
    `meters**2`
  - `bmi = ( kilos /` `# Python "knows" ...`
    `meters**2 )` `# ...it's continued`

- (This behavior seems to have changed slightly with the most recent versions of the language.)

next up...

- compound types – *collections*
- zyBooks chapter 3, "Types"