Data Types in Python

• In Python and other programming languages, different kinds of data are stored and manipulated differently.

• A data type specifies:
  • a domain: a set of possible values (as in SQL)
  • a set of operations that can be performed on values of that type

• In Python, there are two main data types for numbers:
  • integers: int
  • numbers that can have fractional parts (floating-point numbers): float
    • like the REAL type in SQL
    • used whenever a value has a decimal point
Data Types in Python (cont.)

• To determine the type of a value, we can use the built-in \texttt{type()} function.
  
  \textbf{examples:}
  
  >>> \texttt{type(12)}
  \texttt{<class 'int'>}
  
  >>> \texttt{type(3.1495)}
  \texttt{<class 'float'>}
  
  >>> \texttt{type(3.0)}
  \texttt{<class 'float'>}
  
  >>> \texttt{type(2 ** 100)}
  \texttt{<class 'int'>}
  
  >>> \texttt{foo = 100}
  
  >>> \texttt{type(foo)}
  \texttt{<class 'int'>}
  
  >>> \texttt{type(input)}
  \texttt{<class 'builtin_function_or_method'>}

Data Types and Variables

• Variables in Python do \textit{not} have a type, so it's acceptable to change the type of the value assigned to a variable.
  
  \textbf{examples:}
  
  >>> \texttt{value = 3.1495}
  
  >>> \texttt{print(value)}
  \texttt{3.1495}
  
  >>> \texttt{value = 5280}
  
  >>> \texttt{print(value)}
  \texttt{5280}
  
  >>> \texttt{value = 'Hello world!'}
  
  >>> \texttt{print(value)}
  \texttt{Hello world!}
Different Data Types Have Different Operations

• Recall the list of operators for numbers:
  + addition
  - subtraction
  * multiplication
  / division
  ** exponentiation
  % modulus: gives the remainder of a division
    example: \(11 \% 3\) evaluates to 2

• There are really two sets of operators: one set for \texttt{int} values, and one for \texttt{float} values.

• 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

Two Types of Division

• One exception to the rules on the last slide: the regular division operator (/) \emph{always} produces a float result, regardless of whether the operands are ints or floats.
  • examples:
    >>> 5 / 3
    1.6666666666666667
    >>> 6 / 3
    2.0
    >>> 11.0 / 5
    2.2

• The / operator is sometimes called the \emph{float division} operator.
Two Types of Division (cont.)

- Sometimes, it’s useful to perform integer division, which discards the fractional part of the result (i.e., everything after the decimal).

- In Python 3, there is a separate `//` operator for integer division.
  - examples:
    ```python
    >>> 5 // 3
    1
    >>> 6 // 3
    2
    >>> 11 // 5
    2
    ```
  - Note that the `//` operator truncates anything after the decimal. It does not round.

Using the Division Operator

- Recall our change-adder program:
  ```python
  quarters = eval(input("number of quarters? "))
dimes = eval(input("number of dimes? "))
nickels = eval(input("number of nickels? "))
pennies = eval(input("number of pennies? "))

  cents = quarters*25 + dimes*10 + nickels*5 + pennies
  print("you have", cents, "cents")
  ```

- Let’s change it to print the result in dollars and cents.
  - for example, 327 cents would print as 3 dollars, 27 cents
    ```python
    cents = quarters*25 + dimes*10 + nickels*5 + pennies
dollars = ______________________
cents = ______________________
print("you have", dollars, "dollars and", cents, "cents")
```
Type Conversions

- There are built-in functions for converting to any numeric type:
  - `float(n)`: converts n to a float
  - `int(n)`: converts n to an int, discarding any fractional part

Examples:
>>> int(8.72532)
8
>>> float(8)
8.0
>>> float(2**60)
1.152921504606847e+018

Type Conversions (cont.)

- Using a type-conversion function does not change the type of the value stored in memory.
  - examples:
    >>> measurement = 3.7
    >>> int(measurement)
    3
    >>> measurement
    3.7

- How could we change the type of the value stored in memory?
  - this works because variables do not have a type
Rounding a Number

- `round(n)` rounds the number n to an integer:
  ```python
g>>> round(8.5)
  9
  g>>> round(8.49)
  8
  g>>> round(2.8)
  3
  ```

- `round(n, d)` rounds the number n to d places after the decimal. If n is a float, it remains a float. If n is an int, it remains an int.
  ```python
g>>> round(8.7583, 2)
  8.76
  g>>> round(8.7583, 1)
  8.8
  g>>> round(8, 2)
  8
  g>>> round(10.595, 2)
  10.6
  ```

  (note that non-essential 0s are not displayed)

Python's Math Module

- Python comes with a math module that contains definitions for a number of mathematical functions and constants, including:
  - `sqrt(n)`: computes the square root of a number
  - trigonometric functions: `sin(n), cos(n), tan(n)`
  - constants: `pi, e`

- To use them, we need to:
  - `import` the module
  - prepend the name of the module (e.g., `math.sqrt(25)`)

  ```python
g>>> math.sqrt(25)  # won't work before import
  NameError: name 'math' is not defined
  g>>> import math
  g>>> math.sqrt(25)
  5.0
  g>>> math.pi
  3.141592653589793
  ```
Height Converter

• Let's design and write a program that reads a height in centimeters and computes:
  • the height in inches (rounded to the nearest inch)
  • the height in feet, which any fraction of a foot expressed in inches

• Example interaction:
  
  Enter your height in cm: 172
  You are 68 inches tall (5 feet, 8 inches).

• Conversion factor: 1 cm = 0.393700787 inches

Height Converter (cont.)

• Pseudocode:

• Python code: