Lecture 3
Input, Output and Data Types

Goals for today
• Variable Types
  • Integers, Floating-Point, Strings, Booleans
  • Conversion between types
  • Operations on types
• Input/Output
  • Some ways of getting input, and formatting output

Variable Types
• Computer memory consists of a bunch of 1s and 0s
• Remember that a variable describes some area of memory
  • That memory contains only 1s and 0s.
• How do we know how to interpret a set of 1s and 0s?
  • The variable type says how we should interpret that data

Types of Variables
• There are some standard types that are built in to the language or commonly used. Some of these include:
  • Integers
  • Floating-Point Numbers
  • Booleans
  • Strings
• We’ll encounter other types later
• There are also ways to create our own types, but we won’t be discussing these.

Integers
• Integers are just that – integers
• Can be positive or negative
• Examples:
  • 1
  • 2
  • 100
  • 0
  • -20
• Probably the most commonly used variable type
Floating-Point Numbers

- Can represent numbers with a decimal point
- Examples:
  - 1.01
  - 2.0
  - 100.0
  - -20.05
  - 0.004
- May be best to think of these as in "Scientific Notation"
  - A Mantissa (the values, assumed to have 1 digit before the decimal point)
  - An Exponent (the power of 10 that the Mantissa is multiplied by)
  - $1234.567 = 1.234567 \times 10^3$

Booleans

- Booleans will have the value True or False
- That’s it!
- We’ll see a lot more use of Booleans next week...

String

- A string is a way of describing text
- Strings are made up of characters “strung” together.
- When we want to define a string, we usually must specify it within quotation marks, otherwise it might look like a variable name(s).
- You can use either single quotes or double quotes
  - "This is a string"
  - 'So is this'

Side note: Strings

- If we specify a string with single or double quotes, how do we include a quotation mark in the string itself?
  - "He said "I'm tired.""
  - "He said "I'm tired.""
- If we want an apostrophe, use double quotes for the string:
  - "It's"
- If we want quotation marks, use single quotes for the string
  - 'He said, 'What?'' to her'
- But what if we want both?

What if we want both ‘ and ” in a string?

- Two choices
  1. Use triple quotes to begin/end string (single or double):
     - '''He said "I'm tired."'''
     - """He said "I'm tired.""
  2. Use the backslash (\) as an “escape” character
     - Put the \, before the character you want to specify
     - 'He said "I\'m tired."
     - "He said \"I'm tired.\"

Knowing a Variable Type

- In Python, the type of a variable is implied based on how it was created or last assigned.
- In many other languages, we must explicitly define what the type of a variable will be.
- In this case, variable a will be an integer:
  - a = 5
- In this case, variable a will be a floating-point number:
  - a = 5.0
- In this case, variable a will be a Boolean:
  - a = True
- In this case, variable a will be a string:
  - a = "Five"
- It is important to know the type of a variable, or you can get strange effects!
What type is it?

- For the following, what is the type of the variable \( x \)?
  - \( x = 2 \)  
  - \( x = 2.0 \)  
  - \( x = "True" \)  
  - \( x = 2 + 3 \)  
  - \( x = 2.0 \times 3.0 \)  
  - \( x = 2 / 3 \)  
  - \( x = 2 / 2 \)

Type and behavior

- Operations on variables/values can vary depending on the type of the operands.
- Let’s look at the basic addition operator: +
- What is output by the following?
  - \( x=1+2 \)
  - \( \text{print}(x) \)

Type and behavior

- Operations on variables/values can vary depending on the type of the operands.
- Let’s look at the basic addition operator: +
- What is output by the following?
  - \( x=1.0+2.0 \)
  - \( \text{print}(x) \)

Output:

- 3

A reminder of output

- Basic output is to the “console” – a window or the screen that shows the output of the program.
- To show the value of a variable, we “print” it
- Command is: \( \text{print}(x) \)
  
  where \( x \) is the thing you want to be printed.
- The \( x \) can be:
  - a constant value (called a “literal”),
  - a variable,
  - or an expression

Type and behavior

- Operations on variables/values can vary depending on the type of the operands.
- Let’s look at the basic addition operator: +
- What is output by the following?
  - \( x=1.0+2.0 \)
  - \( \text{print}(x) \)

Output:

- 3.0

Printing a floating-point value will include the decimal point.
Type and behavior

- Operations on variables/values can vary depending on the type of the operands.
- Let's look at the basic addition operator: +
  
  What is output by the following?
  ```
x = "1" + "2"
p = print(x)
```

Output:
```
12
```

Why did that happen?

- For numerical types, the + operator means addition
- For strings, the + operator means “concatenation”
- Remember, “1” is a string (the text representation), not the integer 1
- So, “1” + “2” is the string “1” joined to the string “2”, giving the result, a string “12”.
  - Notice that when the string is printed, the quotation marks don't appear.

Other operations

- Strings have + defined (concatenation)
  - Needs to be between two strings
  - string + string gives a new string
  - string + int gives an error, though
- How is the operation - defined for strings?
  - It's not! Trying to subtract two strings will give an error
- Not every operation is defined for every type (in fact, most won't be)
- For strings, - and / are not defined at all, * is defined between two strings, and * is defined for a string and an int
- What do you think 3"Howdy" will be?

Converting between types

- It is possible to convert values of one type into a value of another type.
  - Not for every type combination, though
- General format: new_type(value)
  - value is a variable, expression, or literal of some type
  - new_type is the type to convert into
- Example: converting int to a float:
  - float(3) - this becomes the value 3.0
  - x = 2 - x has the integer value 2
  - y = float(x) - y has the float value 2.0, x still has the int value 2
Converting floats to ints

- When converting a floating-point number to an integer, the value is truncated.
- Any fractional portion is dropped off, only the whole portion remains.
- `int(2.0)` has the value 2
- `int(3.14)` has the value 3
- `int(4.9)` has the value 4
- `int(0.01)` has the value 0
- `int(-1.3)` has the value -1
- `int(-1234.56)` has the value -1234

Converting from strings to ints/floats

- Strings, if they “clearly” define an int/float, can be converted to one of those.
- `int('3')` - this has the integer value 3
- `float('3.14')` - this has the floating-point value 3.14
- `float('2')` - this has the floating-point value 2.0
- `int('-2.5')` - this is an error (notice, it does NOT convert to a float, then to an int)

Converting from a number to a string

- The conversion is done with the command `str`.
- Does a direct conversion into a string. Floating-point values always have at least one digit before and after the decimal point.
- `str(1)` has the value '1'
- `str(2.5)` has the value '2.5'
- `str(1/2)` has the value '0.5'
- `str(18*1.0)` has the value '18.0'

Note: There is also a "repr" alternative to "str".

For most types, repr and str work the same, but "repr" lets you convert a string to a string that is printed as shown (with quotation marks, new lines, etc.)

Boolean conversions

- Remember, Booleans have the value True or False.
- When converting FROM a Boolean value
- True is assumed to have the value 1
- False is assumed to have the value 0
- When converting TO a Boolean value
- The numeric value 0 has the value False
- Anything else has the value True

- `int(True)` has the value 1
- `float(True)` has the value 1.0
- `float(False)` has the value 0.0
- `bool(0)` has the value False
- `bool(3)` has the value True
- `bool('0')` has the value True (is not numeric 0)
- `bool('0.0')` has the value True (is not numeric 0)
- `bool('False')` has the value True (is not numeric 0)

Type conversion example

- What do you think the value of this expression is?

```
str(float(str(3/2)+str(int(3/2))))*int(int(str(2)+str(7))/int(10.3))
```
Type conversion example

• What do you think the value of this expression is?

  \[
  \text{str(float(str(3/2)+str(int(3/2))))*int(int(str(2)+str(7))/int(10.3))} \\
  \text{str(float('1.5'+1'))*int(27/10)} \\
  \text{str(float('1.51'))*int(2.7)} \\
  \text{str(1.51)*2} \\
  \text{'1.51'*2} \\
  \text{'1.511.51'}
  \]

Basics of Output with print()

• As we saw earlier, the basic command for output is the print command.
  
  • The print command formats the output in a readable way
    • By default, it also ends the line it prints on, so the next thing printed will be on the next line
    • More than one value can be specified in the parentheses, separated by commas
      • Each thing is printed, separated by a space
        • print(2.0,'is',2) outputs: 2.0 is 2

Printing strings

• Often, to get the format we want, it’s easiest to create a string ourselves, and print the string.
  
  • For example, say we had values stored in variables x and y, and we wanted to print out: "<xvalue>:<yvalue>" (the values, separated by a colon)

  
  x=3
  y=4
  print(x,:',y)
  print(str(x) + ':' + str(y))

  • Notice lack of spaces in second option

Formatting Strings

• There are a lot of options for formatting strings in different ways
  • Often helps to line up data or get exact formatting
  • We’ll see some of these in labs throughout the course

  • For example, we can “pad” a string with extra spaces to the left and/or right, using the .ljust(), .rjust(), and .center() commands immediately after a string

  "2.3".ljust(10) - has value: '2.3       '
  "2.3".rjust(10) - has value: '       2.3'
  "2.3".center(10) - has value: '   2.3    '

Modifying the print() command

• By default, the print command, when printing
  • Separates all the items (listed separated by commas) with a space
  • Ends the line after printing (so next thing appears on next line)

  • We can actually change how the print command handles both of those things!
  • To change how items are separated, write:
    • print(...) sep='<something>'
  • To change what is done after printing, write:
    • print(...) end='<something>'

  • Note that <something> could be an empty string, if you want nothing printed between items or at the end of a print statement

Example: changing print()

• What would this output?

  print("Test",3,5)
  print("Test",3,5,sep='\',) 
  print("Test",3,5,sep='\',end=':')
  print(15)
Example: changing print()

• What would this output?
  ```python
  print("Test",3,5)
  print("Test",3,5,sep=',')
  print("Test",3,5,sep=' ',end=':')
  print(15)
  ```

  ![Console](console.png)
  
  Test 3 5
  Test,3,5

Example: changing print()

• What would this output?
  ```python
  print("Test",3,5)
  print("Test",3,5,sep=',')
  print("Test",3,5,sep=' ',end=':')
  print(15)
  ```

  ![Console](console.png)
  
  Test 3 5
  Test,3,5:

Example: changing print()

• What would this output?
  ```python
  print("Test",3,5)
  print("Test",3,5,sep=',')
  print("Test",3,5,sep=' ',end=':')
  print(15)
  ```

  ![Console](console.png)
  
  Test 3 5
  Test,3,5:
  Test,3,5:15

Formatting numbers

• Often, we want to output numbers with varying degrees of precision
• This turns out to be a bit more complicated – we will come back to this after seeing some more material

Getting input

• We've seen lots of examples of output – the print statement – but what about input?
• We will assume here that our input is coming from a person typing on the keyboard into the main window.
  • This is as opposed to input from a file, or a device, etc.
• This input source is referred to in different ways: "standard input" or input from the "console" are two of the more common ones.
### The `input()` command

- We have a command, `input()`, available to get input.
  - We have to assign `input()` to a variable.
  - E.g., `thing_I_typed = input()`.
- Important: All input comes in as a string.
  - That is, all input from the `input()` command comes in as a string.
  - If we want to input a number, we have to convert the string to a number.
    - E.g., `age = int(input())` will read in what is typed, convert it to an integer, and save it in the variable `age`.
  - Input is read until the end of the line is entered.

### Example: a program to compute area of a circle

```python
from math import *
r = float(input())
print(pi*r**2)
```

- When run, the user will type a number, and the program will output the area of a circle with that radius.
  - Try it out!
  - Note: we have to have the "from math import *" line to have `pi` defined.

### Example: a program to compute area of a circle

```python
from math import *
r = float(input())
print(pi*r**2)
```

### Console

1
3.141592653589793

### What if we want to output something more descriptive?

- How would you change the program, so that it printed a description of the answer?

```python
from math import *
r = float(input())
print("The area of the circle is " + str(pi*r**2))
```

### Console

1
The area of the circle is 3.141592653589793

### What if we want to ask the user for input?

- How would you change the program, so that it asks the user to input a radius?

```python
from math import *
r = float(input())
print("The area of the circle is " + str(pi*r**2))
```

### Console

1
The area of the circle is 3.141592653589793
What if we want to ask the user for input?

• How would you change the program, so that it asks the user to input a radius?

```
from math import *
print("Enter the radius of a circle:")
r = float(input())
print("The area of the circle is " + str(pi*r**2))
```

```
Console
Enter the radius of a circle: 1
The area of the circle is 3.141592653589793
```

What if we want to ask the user for input?

• Another alternative, then, would have been:

```
from math import *
r = float(input("Enter the radius of a circle: "))
print("The area of the circle is " + str(pi*r**2))
```

```
Console
Enter the radius of a circle: 1
The area of the circle is 3.141592653589793
```

Input with a prompt

• The `input()` command can also print out the text to prompt the user itself.
• Writing `input(x)`, where `x` is a literal, variable, or expression, will print out `x` before getting the user’s input.
• Note: there is NOT a new line printed after that prompt is printed

• Example: `user_name = input("Enter your name: ")`

```
Console
Enter your name:  
```

The user typed here