Introduction to the Python language

CS111 Computer Programming

Department of Computer Science
Wellesley College

Canopy helps you edit and execute Python programs

Jupyter notebooks for hands-on activities
E.g.: lecture-02.ipynb for intro to Python

Python Intro Overview [Slide from Tuesday]

- **Values:**
  - 10 (integer),
  - 3.1415 (decimal number or float),
  - 'wellesley' (text or string)

- **Types:** numbers and text: `int, float, str`
  - `type(10)`
  - `type('wellesley')`

- **Operators:** `+ - * / % =`

- **Built-in functions:** `max, min, len, int, float, str, round, print, raw_input`

- **Expressions:** (they always produce a value as a result)
  - `len('abc') * 'abc' + 'def'`

Knowing the type of a value allows us to choose the right operator when creating expressions.
Simple Expressions:
Python as calculator

<table>
<thead>
<tr>
<th>Input expressions</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+2</td>
<td>3</td>
</tr>
<tr>
<td>3*4</td>
<td>12</td>
</tr>
<tr>
<td>3 * 4</td>
<td># Spaces don't matter 12</td>
</tr>
<tr>
<td>3.5 * 2.0</td>
<td># Floating point (decimal) operations 7.0</td>
</tr>
<tr>
<td>2 + 3 * 4</td>
<td># Precedence 14</td>
</tr>
<tr>
<td>(2 + 3) * 4</td>
<td># Overriding precedence with parantheses 20</td>
</tr>
<tr>
<td>11 / 4</td>
<td># Integer division 2</td>
</tr>
<tr>
<td>11.0 / 4.0</td>
<td># Floating point (decimal) division 2.75</td>
</tr>
<tr>
<td>11 / 4.0</td>
<td># Floating point (decimal) division 2.75</td>
</tr>
<tr>
<td>11 % 4</td>
<td># Integer remainder 3</td>
</tr>
</tbody>
</table>

Concepts in this slide:
- numerical values
- math operators
- expressions

Strings and concatenation

A string is just a sequence of characters that we write between a pair of double quotes or a pair of single quotes.

<table>
<thead>
<tr>
<th>Input expressions</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CS111&quot;</td>
<td># Double quotes 'CS111'</td>
</tr>
<tr>
<td>'rocks!'</td>
<td># Single quotes 'rocks!'</td>
</tr>
<tr>
<td>'CS111 ' + 'rocks!'</td>
<td># Concatenation 'CS111 rocks!'</td>
</tr>
<tr>
<td>'111' + 5</td>
<td># Type error TypeError</td>
</tr>
<tr>
<td>'111' + '5'</td>
<td># Concatenation '1115'</td>
</tr>
<tr>
<td>111 + 5</td>
<td># Integer sum 116</td>
</tr>
<tr>
<td>'111' * 5</td>
<td># Repeated concatenation '11111111111111'</td>
</tr>
</tbody>
</table>

Concepts in this slide:
- string values
- string operators

Variables

A variable names a value that we want to use several times in a program. An assignment statement binds a name to a value, declaring in this way the new variable. A suitable model to think of a variable is that of a box that has a label and a value stored inside it.

Note: The symbol = is pronounced “gets” not “equals”!

```
In [...] Out [...]
fav = 17 adv 4 # assignment statement has no output
fav 17
fav + fav  34
lucky = 8  8
fav + lucky 25
aSum = fav + lucky a_sum 25
aSum * aSum 625
fav = 12
fav = fav - lucky
name = 'CS111' name 'CS111'
nname * fav 'CS111CS111CS111CS111'
```

Concepts in this slide:
- variables
- assignment statement
- model
- NameError

Model: Variable as a Box

- Variables are names we make up (but, there are rules for creating these names)
- A variable name should appear for the first time in an assignment statement.
- A value is stored in a “box”.
- The variable “labels” the box.
- When a variable is used in expressions, we lookup for the “box” with that name and read its value.
- We can reassign a (new) value to a box.
- If we use a name in an expression without using it in an assignment first, we get a NameError.
Built-in functions: \texttt{max} and \texttt{min}

Python has many built-in functions, we don’t need to define them, we just use them. Their names are shown in a green color in Canopy. Variable names are black.

\begin{verbatim}
In [...] Out [...]
min(7, 3) 3
max(7, 3) 7
min(7, 3, 2, 9) # can take any num. of arguments 2
smallest = min(-5, 2) # smallest gets -5
largest = max(-3, -10) # largest gets -3
max(smallest, largest, -1) -1
\end{verbatim}

The inputs to a function are called its arguments and the function is said to be called on its arguments. In Python, the arguments in a function call are delimited by parentheses and separated by commas.

Built-in functions: \texttt{len}

When applied to a string, the built-in \texttt{len} function returns the number of characters in the string. This function will throw a \texttt{TypeError} if used with non-string values.

\begin{verbatim}
In [...] Out [...]
len('CS111') 5
len('CS111 rocks!') 12
len('com' + 'puter') 8
course = 'computer programming'
len(course) 20
len(111) # TypeError
\end{verbatim}

Built-in functions: \texttt{type}

Each Python value has a type. It can be queried with the built-in \texttt{type} function. Types are special kinds of values (not strings). Knowing the type of a value is important when writing expressions containing the value.

\begin{verbatim}
In [...] Out [...]
type(111) int
type(4.0) float
type('CS111') str
type('111') str
type(7/4) int
type(7.0/4.0) float
type(7.0/4) int
type(max(7, 3)) int
x = min(7, 3)
type(x) int
phrase = 'CS111' + 'rocks!'
type(phrase) str
type(type(111)) type
\end{verbatim}

Built-in functions: \texttt{str}

The \texttt{str} built-in function returns a string representation of its argument. It is used to create string values from int-s and float-s to use in expressions with other string values.

\begin{verbatim}
In [...] Out [...]
str('CS111') 'CS111'
str(17) '17'
str(4.0) '4.0'
'CS' + 111 TypeError
'CS' + str(111) 'CS111'
len(str(111)) 3
len(str(min(17, 3))) 1
nameLen = len('CS' + str(max(110, 111)))
str(nameLen) '5'
\end{verbatim}

Example of a complex expression. First, max is called, then str, then +, then the function len.
**Built-in functions: int**

When given a string that’s a sequence of digits, optionally preceded by +/-, `int` returns the corresponding integer.

When given a floating point number, `int` truncates it toward zero.

When given an integer, `int` returns that integer.

```python
In [...]  
int('42')  
int('-273')  
123 + '42'  
123 + int('42')  
int('3.141')  
int('five')  
int(3.141)  
int(98.6)  
int(-2.978)  
int(42)  

Out [...]  
42  
-273  
TypeError  
165  
ValueError  
ValueError  
3  
98  
-2  
42
```

**Oddities of floating point numbers**

In computer languages, floating point numbers (numbers with decimal points) don’t always behave like you might expect from mathematics. This is a consequence of their fixed-sized internal representations, which permit only approximations in many cases.

```plaintext
In [...]  
2.1 - 2.0  
2.2 - 2.0  
2.3 - 2.0  
1.3 - 1.0  
100.3 - 100.0  
10.0/3.0  
1.414*(3.14159/1.414)  

Out [...]  
0.1000000000000009  
0.2000000000000018  
0.2999999999999998  
0.3000000000000004  
0.29999999999999716  
3.3333333333333335  
3.1415900000000003
```

**Built-in functions: float**

When given a string that’s a sequence of digits, optionally preceded by +/-, and optionally including one decimal point, `float` returns the corresponding floating point number.

When given an integer, `float` converts it to floating point number.

When given a floating point number, `float` returns that number.

```python
In [...]  
float('3.141')  
float('-273.15')  
float('3')  
float('3.1.4')  
float('pi')  
float(42)  
float(98.6)  

Out [...]  
3.141  
-273.15  
165  
ValueError  
ValueError  
42.0  
98.6
```

**Built-in functions: round**

When given one numeric argument, `round` returns a floating point version of the integer it’s closest to.

When given two arguments (a numeric argument and an integer number of decimal places), `round` returns the result of rounding the first argument to the number of places specified by the second.

```python
In [...]  
round(3.14156)  
round(98.6)  
round(-98.6)  
round(3.5)  
round(4.5)  
round(2.718, 2)  
round(2.718, 1)  
round(2.718, 0)  
round(1.3 - 1.0, 1)  
round(2.3 - 2.0, 1)  

Out [...]  
3.0  
99.0  
-99.0  
4.0  
5.0  
2.72  
2.7  
3.0  
0.3  
0.3
```
**Built-in functions: **print

`print` displays a character-based representation of its argument(s) on the screen. It does not evaluate to a result value.

**Input statements**

In [...]  
`print(7)`  
`print('CS111')`  
`print('CS' + 111)`  
`print('CS' + str(111))`  
`print(len(str('CS111')) * min(17,3))`  
`college = 'Wellesley'`  
`print('I go to ' + college)`  
`dollars = 10`  
`print('The movie costs ' + str(dollars) + ' dollars.' )`  
`print 1+2, 6*7, 'foo' + 'bar'`

*The final example shows an idiomatic use of print with commas to display multiple values on the same line. Notice the lack of parentheses.*

**Characters displayed in console (not the output value of the expression! )**

7  
CS111  
TypeError  
CS111  
15  
I go to Wellesley  
The movie costs 10 dollars.  
3 42 foobar

**Expression values vs. print**

Notice the field Out[] when the input is a function call, expression, or variable.

The function `print` doesn't output a value, it only displays the result on the screen.

**More built-in functions:**

`raw_input`

`raw_input` displays its argument on the screen and waits for the user to input text, followed by Enter/Return. It returns the entered value as a string.

In [1]: `raw_input('Enter your name: ')`

Enter your name: Phil E. Buster  

Out [1]: 'Phil E. Buster'

*Brown text is prompt.*

In [2]: `age = raw_input('Enter your age: ')`

Enter a number: 19  

Out [3]: '19'

*Variable assignment. No output.*

In [3]: `age`

Out [3]: '19'

*Return value from raw_input is a STRING. Need to be converted to a numerical type as needed.*

In [4]: `age + 4.0`

Out [4]: TypeError

*Example of “nesting” two functions*

In [5]: `age = float(raw_input('Enter your age: '))`

Enter a number: 19

In [6]: `age + 4.0`

Out [6]: 23.0

*1. Give meaningful names to variables.  
2. Use space around operators (e.g., =, +)  
3. Use comments at the top of file  
4. Organize code in “blocks” based on its meaning and provide comments.  
5. Use space between blocks to improve readability.*

**Code Styling Advice**

The script file nameage.py

**Concepts in this slide:**  
the 80-character limit, coding advice.

Notice the vertical bar that marks the 80th character. Don’t write code beyond that line.
Error messages in Python

**Type Errors**
- `'111' + 5`  
  TypeError: cannot concatenate 'str' and 'int' objects
- `5 + '111'`  
  TypeError: unsupported operand type(s) for +: 'int' and 'str'
- `len(111)`  
  TypeError: object of type 'int' has no len()

**Value Errors**
- `int('3.142')`  
  ValueError: invalid literal for int() with base 10: '3.142'
- `float('pi')`  
  ValueError: could not convert string to float: pi

**Name Errors**
- `CS + '111'`  
  NameError: name 'CS' is not defined

**Syntax Errors**
- `2ndValue = 25`  
  SyntaxError: invalid syntax
  - For syntax errors, the arrow ^ doesn't always point exactly to where the error is. In this case, the issue is the number 2 that starts the variable name. It's not allowed to start a variable name with a number.

---

**Test your knowledge**

1. Create simple expressions that combine values of different types and math operators.
2. Which operators can be used with string values? Give examples of expressions involving them. What happens when you use other operators?
3. Write a few assignment statements, using as assigned values either literals or expressions. Experiment with different variable names that start with different characters to learn what is allowed and what not.
4. Perform different function calls of the built-in functions: max, min, len, type, int, str, float, round.
5. Create complex expressions that combine variables, function calls, operators, and literal values.
6. Use the function print to display the result of expressions involving string and numerical values.
7. Write simple examples that use raw_input to collect values from a user and use them in simple expressions. Remember to convert numerical values.
8. Create situations that raise different kinds of errors: Type, Value, Name, or Syntax errors.