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`
- Operators: + - * / % =
- Built-in functions: `max`, `min`, `len`, `int`, `float`, `str`, `round`, `print`, `raw_input`
- Expressions: (they always produce a value as a result)

Knowing the type of a value allows us to choose the right operator when creating expressions.

* In Python 2.7, `print` is actually not a function and is handled specially, but for simplicity we often treat it like a function.
**Simple Expressions:**

Python as calculator

**Input expressions**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Output</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>12</td>
</tr>
<tr>
<td>3.5 * 2.0</td>
<td>7.0</td>
</tr>
<tr>
<td>2 + 3 * 4</td>
<td>14</td>
</tr>
<tr>
<td>(2 + 3) * 4</td>
<td>20</td>
</tr>
<tr>
<td>11 / 4</td>
<td>2</td>
</tr>
<tr>
<td>11.0 / 4.0</td>
<td>2.75</td>
</tr>
<tr>
<td>11 / 4.0</td>
<td>2.75</td>
</tr>
<tr>
<td>11 % 4</td>
<td>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>Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CS111&quot;</td>
<td>'CS111'</td>
</tr>
<tr>
<td>'rocks!'</td>
<td>'rocks!'</td>
</tr>
<tr>
<td>'CS111' + 'rocks!'</td>
<td>'CS111 rocks!'</td>
</tr>
<tr>
<td>'111' + 5</td>
<td>TypeError</td>
</tr>
<tr>
<td>'111' + '5'</td>
<td>'1115'</td>
</tr>
<tr>
<td>111 + 5</td>
<td>116</td>
</tr>
<tr>
<td>'111' * 5</td>
<td>'111111111111111'</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”!

<table>
<thead>
<tr>
<th>Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>fav = 17</td>
<td>17</td>
</tr>
<tr>
<td>fav + fav</td>
<td>34</td>
</tr>
<tr>
<td>lucky = 8</td>
<td>8</td>
</tr>
<tr>
<td>fav + lucky</td>
<td>25</td>
</tr>
<tr>
<td>aSum = fav + lucky</td>
<td>25</td>
</tr>
<tr>
<td>aSum * aSum</td>
<td>625</td>
</tr>
<tr>
<td>fav = 12</td>
<td></td>
</tr>
<tr>
<td>fav = fav - lucky</td>
<td></td>
</tr>
<tr>
<td>name = 'CS111'</td>
<td></td>
</tr>
<tr>
<td>name * fav</td>
<td></td>
</tr>
</tbody>
</table>

**Concepts in this slide:**

- variables
- assignment statement
- model

**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.

- \texttt{min}(7, 3)
- \texttt{max}(7, 3)
- \texttt{min}(7, 3, 2, 9)  \hspace{1cm} # can take any num. of arguments
- smallest = \texttt{min}(-5, 2)  \hspace{1cm} # smallest gets -5
- largest = \texttt{max}(-3, -10)  \hspace{1cm} # largest gets -3
- max(smallest, largest, -1)

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.

- \texttt{len}('CS111')  \hspace{1cm} 5
- \texttt{len}('CS111 rocks!')  \hspace{1cm} 12
- \texttt{len('com' + 'puter')}  \hspace{1cm} 8
- course = 'computer programming'
- \texttt{len(course)}  \hspace{1cm} 20
- \texttt{len(111)}  \hspace{1cm} # TypeError

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.

- \texttt{type(111)}  \hspace{1cm} \texttt{int}
- \texttt{type(4.0)}  \hspace{1cm} \texttt{float}
- \texttt{type('CS111')}  \hspace{1cm} \texttt{str}
- \texttt{type('111')}  \hspace{1cm} \texttt{str}
- \texttt{type(7/4)}  \hspace{1cm} \texttt{int}
- \texttt{type(7.0/4.0)}  \hspace{1cm} \texttt{float}
- \texttt{type(7.0/4)}  \hspace{1cm} \texttt{float}
- \texttt{type(max(7, 3))}  \hspace{1cm} \texttt{int}
- \texttt{x = min(7, 3)}  \hspace{1cm} \texttt{int}
- \texttt{type(x)}  \hspace{1cm} \texttt{int}
- \texttt{phrase = 'CS111' + 'rocks!'}
- \texttt{type(phrase)}  \hspace{1cm} \texttt{str}
- \texttt{type(type(111))}  \hspace{1cm} \texttt{type

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.

- \texttt{str('CS111')}  \hspace{1cm} 'CS111'
- \texttt{str(17)}  \hspace{1cm} '17'
- \texttt{str(4.0)}  \hspace{1cm} '4.0'
- 'CS' + 111
- 'CS' + \texttt{str}(111)
- \texttt{len(str(111))}
- \texttt{len(str(min(17, 3)))}
- \texttt{nameLen = len('CS' + str(max(110, 111)))}
- \texttt{str(nameLen)}  \hspace{1cm} '5'

Example of a complex expression.
First, \texttt{max} is called, then \texttt{str}, then +, then the function \texttt{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.

```
In [...]  
int('42')  42  
int('-273') -273  
123 + '42'  165  
123 + int('42')  165  
int('3.141')  ValueError  
int('five')  ValueError  
int(3.141)  3  
int(98.6)  98  
int(-2.978) -2  
int(42)  42

Concepts in this slide: the function `int`, `TypeError`, `ValueError`.
```

**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.

```
In [...]  
float('3.141')  3.141  
float('-273.15') -273.15  
float('3')  3.0  
float('3.1.4')  ValueError  
float('pi')  ValueError  
float(42)  42.0  
float(98.6)  98.6

Concepts in this slide: the function `float`, `ValueError` (two different kinds).
```

**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.

```
In [...]  
2.1 - 2.0  0.1000000000000009  
2.2 - 2.0  0.2000000000000018  
2.3 - 2.0  0.2999999999999998  
1.3 - 1.0  0.3000000000000004  
100.3 - 100.0  0.29999999999999716  
10.0/3.0  3.3333333333333335  
1.414*(3.14159/1.414)  3.1415900000000003

Concepts in this slide: the function `round`, function call with varying number of arguments.
```

**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.

```
In [...]  
round(3.14156)  3.0  
round(98.6)  99.0  
round(-98.6) -99.0  
round(3.5)  4.0  
round(4.5)  5.0  
round(2.718, 2)  2.72  
round(2.718, 1)  2.7  
round(2.718, 0)  3.0  
round(1.3 - 1.0, 1) # compare prev. slide  0.3  
round(2.3 - 2.0, 1) # compare prev. slide  0.3

Concepts in this slide: the function `round`, function call with varying number of arguments.
```
### 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(len(str('CS111')) * min(17, 3))`
- `college = 'Wellesley'`
- `print('I go to ' + college)`
- `dollars = 10`  
- `print('The movie costs ' + str(dollars) + ' dollars. ' + 'foo' + 'bar')`

### 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.

### Expressions vs. Statements

**Expressions**

- They always produce a value:
  - `10`
  - `10 * 20 - 100/25`
  - `max(10, 20)`
  - `int("100") + 200`
  - `fav + 3`
  - "pie" + " in the sky"

**Expressions are composed of values, operators, variables, functions, and any combination of them.**

**Statements**

- They perform an action (that can be visible, invisible, or both):
  - `print(10)`
  - `age = 19`
  - `paper = Canvas(400, 550, 'yellow')`
  - `paper.add(head)`
  - `print('She is ' + str(age) + ' years old. ')`

### 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]: max(10, 20)`
- `Out[1]: 20`
- `In [2]: 10 + 20`
- `Out[2]: 30`
- `In [3]: message = "Welcome to CS 111"`
- `In [4]: message`
- `Out[4]: 'Welcome to CS 111'`
- `In [5]: print(message)`
- `Welcome to CS 111`
- `In [6]: print(max(10, 20))`
- `20`
- `In [7]: print(10 + 20)`
- `30`

### Concepts in this slide:

- The function `print`, an alternative way of using `print` (last line).
- Expressions, statements
- Expressions vs. Statements
- the function `print`, an alternative way of using `print` (last line).
- the function `raw_input`, converting from string.
### Code Styling Advice

The script file `nameage.py`

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.

### 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

---

### 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.