Literals, Variables, Data Types, Statements and Expressions
Python as Calculator

>>> 2 + 2 # add
4

>>> 2 * 3 # multiply
6

>>> 3**2 # powers
9

>>> 12 % 11 # modulo (remainder)
1

Integers
(whole numbers)
Integer Division

>>> 6/2 # division
3

>>> 5/2 # integer division, no decimals
2

>>> 4*(1/4)
Careful
0
Float Division

```python
>>> 6.0/2.0 # division
3.0

>>> 5.0/2.0
2.5

>>> 4.0*(1.0/4.0)
1.0
```
Literals (are fixed values)

• Numbers
  • **Integers** are natural numbers: ..., -2, -1, 0, 1, 2, ... (32 bits)
  • **Floats** contain decimals: 4.5, 67.444443335, 7E2...
  • **Booleans**: True, False
  • **Long** int’s that exceed 32 bit capacity
  • **Complex** numbers: 4.5, 1j * 1j = -1 + 0j

• Strings
  • **Strings** are used to represent words, text, and characters
  • examples (can use single, double or triple quotes):
    • “I am learning python.“
    • 'hello.'
Variables (NAME = VALUE)

• are names we assign to chunks of memory

• we retrieve the values by calling its name

\[
\text{student_id} = 42
\]
\[
\text{print student_id}
\]

• = is the assignment operator, assigns a data value to a variable
Variables Syntax Rules

• Variable names must begin with a letter (uppercase or lowercase) or underscore (_)
  • *good programming convention: variables should not start with uppercase letters, commonly used for something else

• remainder of name can contain letters, (_), and numbers

• names may not contain spaces or special characters

• names are case sensitive and must not be a reserved python keyword
  • myVariable and myvariable refer to different data
Assignment

• other things can be assigned to names

• like functions (from Robot example)

    \texttt{avance = move}

    \texttt{avance() \# calls the same code in move()}

• later we will learn of other things we can name
Statements and Expressions

• Statements perform a task; do not return a value

```python
x = 2
y = 3
print y
```

• Expressions return a value

```python
>>> x + y
6
```
Expressions (evaluate to values)

- Math expressions
  
  >>> 10 * 2 + 3
  23
  
  >>> 10 * (2.0 + 3)
  50.0

- Operator Precedence (PEMDAS)
  
  - Parentheses
  - Exponentiation
  - Multiplication and Division
  - Addition and Subtraction
Expressions (evaluate to values)

• Boolean expressions

  >>> 10 < 2
  False
  >>> 10 >= 10
  True

• Combined with Logical Operators

  >>> (10 < 2) and (10 == 10)
  False
Expressions (evaluate to values)

• Can combine

\[(a \times c + d) > (d \times a - c)\]

• String expressions

```python
>>> "Hel" + "lo"
'Hello'
>>> "Hi"*3
'HHiHiHi'
```

• Later lectures will reveal more about this
# Operator Precedence (top-to-bottom)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Parenthesis (grouping)</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>+x, -x</td>
<td>Positive, Negative</td>
</tr>
<tr>
<td>*, /, %</td>
<td>Multiplication, Division, Remainder</td>
</tr>
<tr>
<td>+, -</td>
<td>Addition, Subtraction</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=, !=, ==</td>
<td>Comparisons</td>
</tr>
<tr>
<td>not x</td>
<td>Boolean NOT</td>
</tr>
<tr>
<td>and</td>
<td>Boolean AND</td>
</tr>
<tr>
<td>or</td>
<td>Boolean OR</td>
</tr>
</tbody>
</table>
Data Types

- Finding out a data type

```python
>>> x = 2.0
>>> print type(x)
<type 'float'>
>>> print type('hi')
<type 'str'>
```
Data Types

• What if data types don’t match?

```python
>>> x = 2
>>> y = x + 3.0 # add an int and float
>>> print type(x)
<type 'int'>
>>> print type(y)
<type 'float'>
```

• **STRONG TYPES** no automatic conversion (for non number types)

```python
>>> x = "str" + 2
TypeError: cannot concatenate 'str' and 'int' objects
```
Data Types

• Explicit conversion

```python
>>> x = "5"
>>> y = 5
>>> x + str(y) # str() converts to str
'55'
>>> int(x) + y # int() converts to int
10
>>> float(x) + y # converts to float
10.0
```
Python Keywords, User Input
Keywords

- RESERVED: do not use as variable names

<table>
<thead>
<tr>
<th>and</th>
<th>as</th>
<th>assert</th>
<th>break</th>
<th>class</th>
<th>continue</th>
</tr>
</thead>
<tbody>
<tr>
<td>def</td>
<td>del</td>
<td>elif</td>
<td>else</td>
<td>except</td>
<td>exec</td>
</tr>
<tr>
<td>finally</td>
<td>for</td>
<td>from</td>
<td>global</td>
<td>if</td>
<td>import</td>
</tr>
<tr>
<td>in</td>
<td>is</td>
<td>lambda</td>
<td>not</td>
<td>or</td>
<td>pass</td>
</tr>
<tr>
<td>print</td>
<td>raise</td>
<td>return</td>
<td>try</td>
<td>while</td>
<td>with</td>
</tr>
<tr>
<td>yield</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
User Input

• Create interactive programs by requesting user input

```python
>>> x = raw_input("Give me a number: ")
Give me a number: 34.5
>>> print x
'34.5'
>>> float(x)
34.5
```
Control Structures
Branching / Conditional Statements

- Decision Making

```
is front of robot clear?

False -> Stop()

True -> Move()
```
if - statement

if front_is_clear():
    move()

if TEST:
    DO THIS
if - else

if front_is_clear():
    move()
else:
    turn_off()

if x % 2 == 0:
    print x, "is even"
else:
    print x, "is odd"
if - elif - else

• If one test fails, perform next test

```python
choice = raw_input("make a choice (a, b, or c): ")
if choice == 'a':
    function_a()
elif choice == 'b':
    function_b()
elif choice == 'c':
    function_c()
else:
    print "Invalid choice."
```
Nested if statements

if x == y:
    print x, "and", y, "are equal"
else:
    if x < y:
        print x, "is less than", y
    else:
        print x, "is greater than", y