Variables and Memory

What’s a variable, really?
What’s a type?

What’s a variable?

First try:
A variable is a *name* for a location in memory (a memory “cell”).

(not quite right ... but it's a start)

When I write “x = 32” I mean “put the value 32 in the memory location named ‘x’.”

Not like variables in math

In math, \( x = x + 1 \) is just wrong

In Python, \( x = x + 1; \) has a meaning ...

Take the value of \( x \),
add 1 to it,
stored the result in \( x \).

What’s memory, really?

Memory cells are one big list, numbered from zero.
The computer access them by “address” (number).

Variable ‘x’ might mean cell #6.
x = 33 might mean:
put 100001 in cell #6
Memory

An address (location) is sent on some of these connectors
Contents go in or out on some of these connectors

01100010₂

Means $62_{16}$ (interpreted as an integer)
Or $98_{10}$
Or ‘b’ (interpreted as an ASCII character)
Or BOUND (interpreted as an x86 instruction)

Or ... it doesn’t “mean” anything, but we can interpret it several ways, as data or as a program instruction.

Memory Representation

It’s all binary (1’s and 0’s)

What does 01100010₂ mean?

(Trick question ... why?)

Where’s the type?

Dynamic Types
(Python, …)

Static Types
(Java, …)

\[
\begin{align*}
\text{x} & = 5 \\
\text{int x;} & \\
\text{x} & = 5;
\end{align*}
\]

\[
\begin{align*}
\text{x} & \quad \text{int} \quad 00000101 \\
\text{int} & \quad \text{x} \quad 00000101
\end{align*}
\]
Dynamic types allow ...

\[ x = 5 \]
\[ x = 5.0 \]
\[ x = "Text" \]

Some basic Python types

<table>
<thead>
<tr>
<th>name</th>
<th>meaning</th>
<th>example (literal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>whole number, represented as 32 or 64 bits*</td>
<td>42</td>
</tr>
<tr>
<td>float</td>
<td>floating point (approximation of real number), typically 64 bits</td>
<td>42.0</td>
</tr>
<tr>
<td>string</td>
<td>sequence of characters (text)</td>
<td>&quot;Hello World&quot;</td>
</tr>
<tr>
<td>boolean</td>
<td>true or false (represented as integers 1 and 0)</td>
<td>True</td>
</tr>
</tbody>
</table>

*usually ... but big integers have a different representation (There are more ... these will be enough for a while)
Operations depend on types!

15 + 32 is 47

“15” + “32” is “1532”

What is it?

x = 15
y = 2
w = x / y;
z = x // y;

What value is in w?
(the answer was different for Python 2)

What is it?

x = 15.0;
y = 2.0;
z = x / y;

What value is in z?

% is the “remainder” or “modulo” operation when applied to a pair of integers
Coercion

“Coercions” are implicit conversions from one type to another

```python
x = 42
print(x)  # What does it print?
x = x + 0.0
print(x)  # Now what does it print?
```

Use cautiously ... can be confusing

Casts (explicit conversions)

Like coercions, but explicit ...

```python
x = 42.84
y = int(x)  # Convert the value of x to integer
             # and then store in y
print(y)
```

What does it print?

Types in Assignment 1

The skeleton code contains this text:

```python
## Get pin code from command line
import sys
if (len(sys.argv) > 1):
    pincode = int(sys.argv[1])
```

What’s going on here?

sys.argv: Command line arguments

The skeleton code contains this text:

```python
## Get pin code from command line
import sys
if (len(sys.argv) > 1):
    pincode = int(sys.argv[1])
```

A list, e.g., [“alphacode.py”, “4293”]
Get the value ... as text

The skeleton code contains this text:

```python
## Get pin code from command line
import sys
if (len(sys.argv) > 1):
    pincode = int(sys.argv[1])
```

A list, e.g., [“alphacode.py”, “4293”]

Then we can work with it ...

The skeleton code contains this text:

```python
## Get pin code from command line
import sys
if (len(sys.argv) > 1):
    pincode = int(sys.argv[1])
```

Now pincode is 4293 (an integer)

Convert to an integer

The skeleton code contains this text:

```python
## Get pin code from command line
import sys
if (len(sys.argv) > 1):
    pincode = int(sys.argv[1])
```

A list, e.g., [“alphacode.py”, “4293”]

Convert to integer ... what really happened?

```python
x = int("42")
```

```
0000 0000 0000 0000 52_10 34_16 50_10 32_16
0011 0010 0011 0010
'4'       '2'

(string) 0011 0100 0011 0010 0000 0000 0000 0000
```
\[ 52_{10}, 34_{16} \quad 50_{10}, 32_{16} \]

\[ '4' \quad '2' \]

(string) 0011 0100 0011 0100 0000 0000 0000 0000

(int) 0000 0000 0000 0000 0000 0000 0000 0000

\[ (\text{multiply by 10 before adding next digit}) \]

\[ 52_{10}, 34_{16} \quad 50_{10}, 32_{16} \]

\[ '4' \quad '2' \]

(string) 0011 0100 0011 0100 0000 0000 0000 0000

(int) 0000 0000 0000 0000 0000 0000 0010 1000

\[ 40 \]

\[ 52_{10}, 34_{16} \quad 50_{10}, 32_{16} \]

\[ '4' \quad '2' \]

(string) 0011 0100 0011 0100 0000 0000 0000 0000

(int) 0000 0000 0000 0000 0000 0000 0010 1000

\[ 42 \]
### Convert to integer ... what really happened?

$x = \text{int}("42")$

```
52_{10}, 34_{16}    50_{10}, 32_{16}
4'                2'
```

(string) \[0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000\]

```
   0011 0010
   0011 0100
   0010 1010
   0000 0000
```

(int) \[0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000\]

$x$

```
   (int) \[0000 0000 0000 0000 0000 0000 0000 0000\]
```

(int) \[0010 1010\]

42

### Variable names matter

The computer doesn’t care ... but other programmers (and graders!) do

“dollars” and “cents” are better than “x” and “y”

but_this_is_not_really_a_very_good_name

longer ≠ better

### Python Naming Conventions

Note: Java naming conventions differ!

```python
## Constants (set once, never change)
MILE_PER_KM = 0.621371192

## Local variables
life_universe_etc = 42
author = "Douglas Adams"

## function
def double( n ) :
    """Always document your functions""
    return n + n
```

### Summary: What’s a variable?

It names a location in memory
Either the variable has a type (e.g., in Java), or the value in the variable has a type (in Python); the type determines how it is interpreted and the meaning of operations (e.g., +) on it.

Variable names should be descriptive (enough) and follow naming conventions

Naming conditions vary among programming languages, and among organizations.