Variables and Memory

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

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,
store the result in \( x \).

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 ‘\( x \)’.

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0101101110</td>
</tr>
<tr>
<td>1</td>
<td>0101101110</td>
</tr>
<tr>
<td>2</td>
<td>0101101000</td>
</tr>
<tr>
<td>3</td>
<td>1101101110</td>
</tr>
<tr>
<td>4</td>
<td>0101110110</td>
</tr>
<tr>
<td>5</td>
<td>0101101110</td>
</tr>
<tr>
<td>6</td>
<td>0111101011</td>
</tr>
<tr>
<td>7</td>
<td>0101110110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Memory

An address (location) is sent on some of these connectors.

Contents go in or out on some of these connectors.

01100010₂

Means 62₁₆ (interpreted as an integer)
Or 98₁₀
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, ... )

x = 5

Static Types
(Java, ... )

int x;

x = 5;

int x

00000101

00000101
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!

What is $15 + 32$?

What is “15” + “32”? 

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

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

Use cautiously ... can be confusing

Casts (explicit conversions)

Like coercions, but explicit ...
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:

## 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:

## 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")
```

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

(string) 0011 0100 0011 0010 0000 0000 0000 0000
\begin{align*}
\text{52}_{10}, 34_{16} & \quad 50_{10}, 32_{16} \\
\text{'4'} & \quad \text{'2'} \\
\end{align*}

\text{(string)} \quad \begin{array}{cccccccc}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}

\begin{align*}
\text{(int)} & \quad 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
& \quad 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
& \quad 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
& \quad 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{align*}

\text{(multiply by 10 before adding next digit)}
Convert to integer ... what really happened?

\[ x = \text{int}("42") \]

```
52  34  50  32
    '4'    '2'
(\text{string}) 0011 0010 0000 0000 0000 0000
```

\[ x \Rightarrow \text{(int)} 0000 0000 0000 0000 0000 0010 1010
\]

```
\text{42}
```

Variable names matter

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

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

\[ \text{but\_this\_is\_not\_really\_a\_very\_good\_name} \]

longer ≠ better

Python Naming Conventions

\textbf{Note: Java naming conventions differ!}

### Constants (set once, never change)

\texttt{MILE\_PER\_KM = 0.621371192}

### Local variables

\texttt{life\_universe\_etc = 42}

\texttt{author = “Douglas Adams”}

### function

\texttt{def double( n ) :}

\texttt{\quad ““Always document your functions””““}

\texttt{\quad 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

\textit{Naming conditions vary among programming languages, and among organizations.}