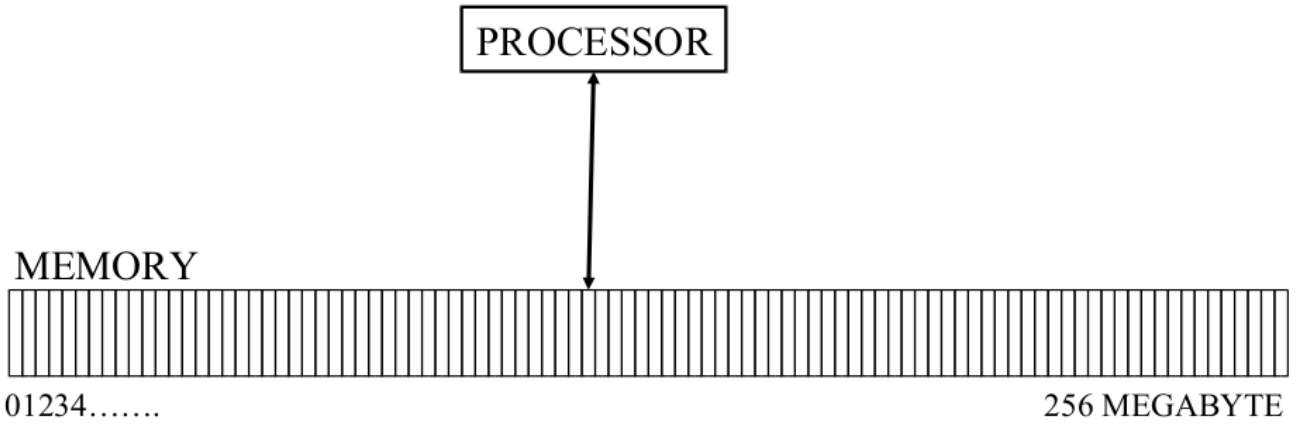
Programming cont’d

IC152 Lecture 6
Feb 2021

The computing machine



The computer has a *processor* and a *memory*. The memory is a series of *locations* to store information.

Slide from IC152, 2018

A snippet of Python code

Python code:

```
a = 1
b = 2
c = a + b
d = c * b
```

Memory:

Name	Cell	*Address*
	?	180
d	6	160
c	3	140
b	2	120
a	1	100
	?	80

= is assignment, not equality

? is a random value

Slide from IC152, 2018

Python automatically allocates memory for a variable when it is first used

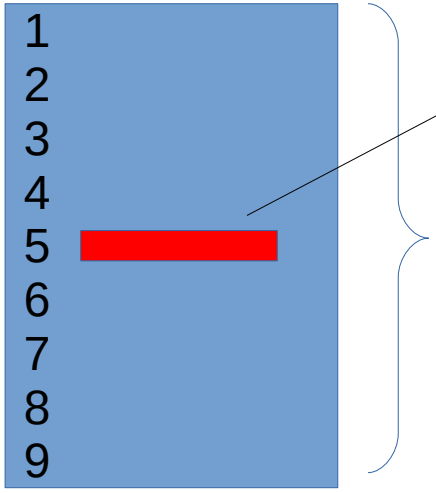
This is not the case for C, C++, Java
Every variable must be declared before using it

Memory is freed when the variable is no longer needed:
Garbage Collection

Programmer (almost) never needs to know the address of a variable

Python is an interpreted language
Execution happens line by line

Pic from Downey's book

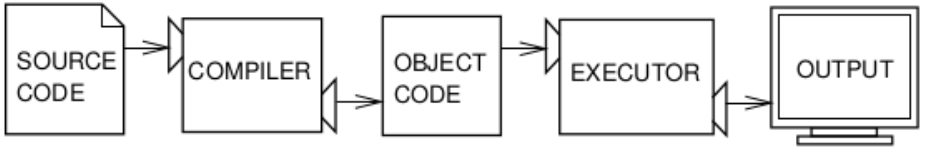


Syntax error in line 5

code

Interpreter executes till line 4 and then gives an error

Languages like C and C++ are *compiled*
The C compiler converts the program into an *executable*



Pic from Downey's book

Python Data Types

- Many built-in types: numerics, text, boolean, …
- Type of `var` determined by "`var = value`"
- Type of `var` can change during execution

```
x = 23        # type of x is integer
x = 2.3       # type of x changes to float
x = 5 + 9j    # x is now complex
```

- Operators:

```
+ - * /
** – power     2**3 → 8
// – round     3/2 → 1.5        3//2 → 1
%  – modulus   10%7 → 3         10%3 → 1
```

Slide from IC152, 2018

Operators & Expressions

- Expression with multiple operators:
  - usually left-to-right, high precedence operators first
- `3 + 4 - 5 → 2` but `3 + 4 * 5 → 23`
- Use parentheses: `(3 + 4) * 5 → 35`
- Comparison and logical operators:

```
3 < 4 → True          3 >= 4 → False
3 < 4 and 5 >= 4 → True and True → True
3 < 4 and 5 == 4 → True and False → False
```

- Numerics have logical values:

```
0 → False     not 0 → True     not 3 → False
```

Slide from IC152, 2018

Operator Precedence

- Precedence table of many Python operators
- High to Low
- Value of

`2**3**2` 64 or 512?

`8/2*4` 1 or 16?

Operators	Associativity
`(), [], {}`	Left-right
`func(), array[]`	Left-right
`**`	Right-left
`*, /, //, %`	Left-right
`+, -`	Left-right
`<, <=, >, >=, !=, ==`	Left-right
`not`	Left-right
`and`	Left-right
`or`	Left-right

6.16 in https://docs.python.org/3/reference/index.html

Slide from IC152, 2018

Type Conversion

- An expression with at least one float → float
- An expression with all integers → float if any `/` results in a fraction, otherwise integer
- `x = 3.5;  int(x)` → 3
- `float(int(x))` → 3.0
- `c = 4 + 3.5j` # `c` is a complex variable
- `c.real` → 4 `c.imag` → 3.5

Integer: unlimited digits
Float: ~15 digits, 10^{-300}..10^{+300}

Strings

- string is a compound type

```
In [4]: x = 'IIT'

In [5]: type(x)
Out[5]: str

In [6]: y = '17'

In [7]: type(y)
Out[7]: str

In [8]: yi = int(y)

In [9]: x_i = int(x)
Traceback (most recent call last):

  File "<ipython-input-9-c06e6edf0465>", line 1, in <module>
    x_i = int(x)

ValueError: invalid literal for int() with base 10: 'IIT'
```

Strings are made up of characters, and are enclosed in ‘ ‘ (or “ ”)
We might want to access its parts, or consider it in whole.

```
In [10]: x
Out[10]: 'IIT'

In [11]: x[0]
Out[11]: 'I'

In [12]: x[1]
Out[12]: 'I'

In [13]: x[2]
Out[13]: 'T'
```


Access the individual elements using an index

Index starts from 0

```
In [15]: fruit = 'banana'

In [16]: len(fruit)
Out[16]: 6

In [17]: fruit[len(fruit)-1]
Out[17]: 'a'
```

`len()` is a built-in function

Strings can be traversed with loops

for loop can also be used

```
 8
 9 fruit = 'banana'
10 index = 0
11 while index < len(fruit):
12     letter = fruit[index]
13     print(letter)
14     index = index + 1
```

while loop

```
17 for c in fruit:
18     print(c)
```

```
b
a
n
a
n
a
```

What will be the value of index?

HW: Accept a string and print it in reverse

```
prefixes = "JKLMNOPQ"
suffix = "ack"

for letter in prefixes:
    print (letter + suffix)
```

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

Slicing strings

```
In [23]: s = "Peter, Paul, and Mary"

In [24]: print(s[0:5])
Peter

In [25]: print(s[7:11])
Paul

In [26]: print(s[17:21])
Mary
```

```
In [28]: fruit = 'banana'

In [29]: fruit[:3]
Out[29]: 'ban'

In [30]: fruit[3:]
Out[30]: 'ana'

In [31]: fruit[:]
Out[31]: 'banana'
```

```
In [32]: 'Zebra' < 'Banana'
Out[32]: False

In [33]: 'Zebra' < 'banana'
Out[33]: True
```

Strings can be compared

```
In [34]: x = 'Zebra' < 'Banana'

In [35]: type(x)
Out[35]: bool
```

What type is `x`?

Strings are immutable

```
greeting = "Hello, world!"
greeting[0] = 'J'
```

Error!

You can create a new string

```
greeting = "Hello, world!"
newGreeting = 'J' + greeting[1:]
print(newGreeting)
```

Concatenate new character to a slice of the original