U.S. INTERNATIONAL 中文 ESPAÑOL

# The Python Times

Tuesday, February 2, 2021 | Today's Paper | Video | 71°F

## Three-eyed aliens causing most computer troubles...

Alien Attack?! Picobot programmer G. Kuenning was subjected to a bizarre encounter with **three-eyed aliens yesterday**. The trinocular tourists, it seems, were conducting experiments that would help them understand "**how humans think**."

The aliens apparently used a shrinking ray that let them enter the programmer's head to see what was happening. A witness reports **deeply disappointed voices** emanating from within.

To escape the attack, Kuenning had to turn the ray on himself. As he shrank, the aliens quickly flew off, departing so fast that he was **unable to use the reverse ray** before they left. "I don't mind," Kuenning mused. In fact, this might help me hide from students tomorrow." see three-eyed alien attack, p. 42

**Composite sketch of one of the attackers, drawn from three-eyewitness accounts**

Today's whether:
if, elif, or else!

**Homework #2**

0) Reading + response
1) **Lab**: *data*
2) **Lab**: *functions*
3) The *fun* in *fun*ctions!

Read sections 2.3-2.5

---

learning a language ~ ***syntax***

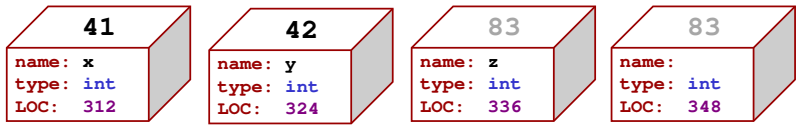
unavoidable, but not the point

... but learning CS ~ ***semantics*** ➡

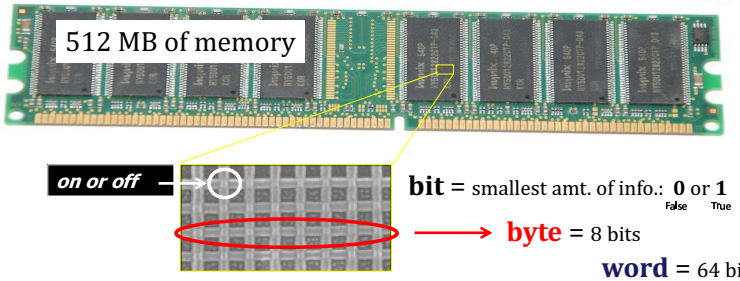
learning how machines ***think***!

---

# Memory! Random Access Memory




a big list of boxes, each with a name, type, location, and value




**bit** = smallest amt. of info.: **0** or **1** (False / True)

**byte** = 8 bits

**word** = 64 bits

---

# All languages use ***data types***

| Type | Example | What *is* it? |
|---|---|---|
| `float` | `3.14` or `3.0` | numeric values with a fractional part, *even if the fractional part is .0* |
| `int` | `42` or `10**100` | integers – Python has *infinite-precision ints!* |
| `bool` | `True` or `False` | T/F results from a test or comparison: `==, !=, <, >, <=, >=` |

"Boolean values"

"Boolean operators"

Hey! Someone can't spelle !

George Boole

**type( x )**

# Python operators

higher precedence

| | |
|---|---|
| parens | `( )` |
| power | `**` |
| negate | `-` |
| times, mod, divide | `* / % //` |
| add, subtract | `+ -` |
| compare | `> == <` |
| assign | `=` |

It's hard to remember all these `%+/*` things! At first, prefer parentheses over precedence.

# `%` the ***mod*** operator

```
7 % 3
8 % 3
9 % 3
30 % 7
```

**`x % y`** is the ***remainder*** when **`x`** is divided by **`y`**

For what values of **`x`** are these **`True`**?

```
x % 2 == 0
x % 2 == 1
x % 4 == 0
x % 4 == 3
```

If x is a year, what happens on these years!?

What happens on these years, football-wise!?

# `//` *integer* division

```
7 // 3
8 // 3
9 // 3
30 // 7
```

**`x//y`** is **`x/y`**, *rounded down* to an integer

*Decomposition of 31 into 7's:*

31 == (4) * 7 + (3)

*But why?*

*Decomposition of x into y's:*

```
x == (x // y) * y + (x % y)
```

*# of full y's in x*

*remainder after "taking" all of the full y's in x*

# the "equals" operators

`=` `!=` `==`

**This is true – *but what is it saying!?***

# Inside the machine…

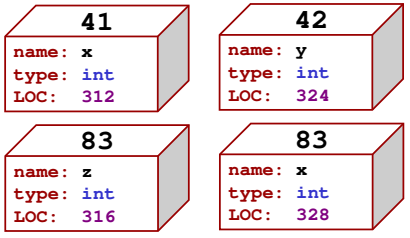
```
x = 41
y = x + 1
z = x + y
x = x + y
```

What's happening in python:

What's happening behind the scenes (in memory):

Computation

Memory (Data Storage)




id, del

# strings: *textual* data

strings

```
s = 'scripps'
c = 'college'
```

type… `type(s)`

len `len(s)`

add! `s + c`

multiply!! `2*s + 3*c`

What did you say!?!

# *Lists* ~ ordered collections of *any* data

string

```
L = [3.14, [2, 40], 'third', 42]
```

| `len(L)` | `L[0]` | `L[0:1]` |
|---|---|---|
| **top-level length** | **indexing** | **slicing** |
| *only counts **top-level** elements* | ***could** return a different type* | ***always** returns the same type, and **always** returns a **sub**structure!* |

# **Indexing** uses `[ ]`

```
s = 'harvey mudd college'
    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

index

`s[0]` is `'h'`

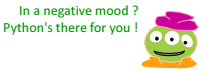
Read as "s-of-zero" or "s-zero" or "s-sub-zero"

`s[17]` is

`s[6]` is

`s[ ]` is `'e'`

## *Negative* indices…




```
          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
s  =  'harvey mudd college'
        -19 -17 -15 -13 -11 -9  -7  -5  -3  -1
           -18 -16 -14 -12 -10 -8  -6  -4  -2
```

Negative indices count ***backwards*** from the end!

`s[-1]` is `'e'`

`s[-18]` is

`s[-7]` is

`s[-0]` is

## *Slicing*

```
s  =  'harvey mudd college'
      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

`s[ start : end+1 ]` ***slices*** the string, returning a substring

first index is the first character

second index is **ONE AFTER** the last character

a missing index means ***that end*** of the string

`s[0:6]` is `'harvey'`

`s[12:18]` is `'colleg'`

`s[17:]` is `'ge'`

`s[:]` is `'harvey mudd college'`

## *Slicing*

```
          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
s  =  'harvey mudd college'
      -19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9  -8  -7  -6  -5  -4  -3  -2  -1
```

What are these slices?

`s[15:-1]` is

`s[:2]` is

and these?

is `'mud'`

is `'e'`


## Skip Slicing

`s[ start : end+1 : ]` the third index is the **stride length**

default is +1

```
s  =  'harvey mudd college'
      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

`s[0:8:2]` is `'hre '`

`s[17:12:-1]` is

`s[::-1]` is

is `'doe'`

`s[1::6]` is

# Skip Slicing

`s[ start : end+1 : ]` ← the third index is the **stride length**

default is +1

```
s = 'harvey mudd college'
    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

`s[0:8:2]` is `'hre '`

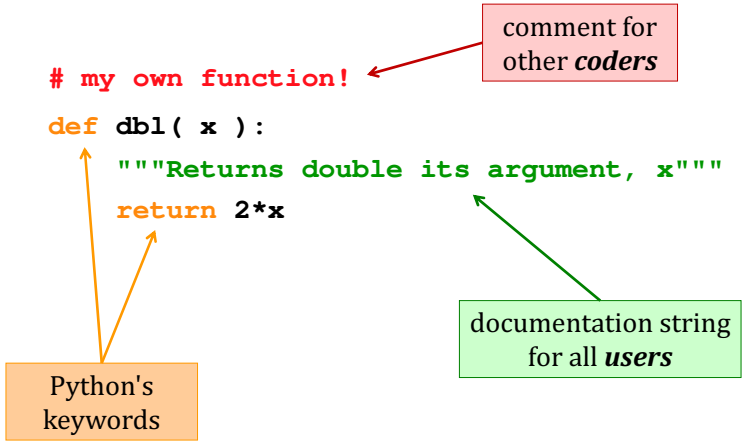
`s[17:12:-1]` is

`s[::-1]` is

is `'doe'`

`s[1::6]` is

# ***Function***ing in Python




Some of Python's *baggage*...

# Function *Fun* !

```
def undo(s):
    """This "undoes" its argument, s"""
    return 'de' + s
```

---

```
>>> undo('caf')
'decaf'
>>> undo(undo('caf'))
```

*strings, lists, numbers ...*
*all **data** are fair game*

# how = works

# *"Quiz"*

name(s)

Run these lines

```
x = 41
y = x + 1
z = x + y
```

What are **x**, **y**, and **z** at this time?

| x | y | z |
|---|---|---|
| | | |

Then run this line

```
x = x + y
```

What are **x**, **y**, and **z** at this time?

| x | y | z |
|---|---|---|
| | | |

*Extra!*

```
a = 11 // 2
b = a % 3
c = b** a+b *a
```

What are the values of **a**, **b**, and **c** after the 3 lines, at left, run?

| a | b | c |
|---|---|---|
| | | |