Python

An interpreter language, so it's cross platform.

Variables

Really easy to assign, simply declare and assign a value

Syntax: foo = "bar", 1, 1.22, etc.

Sequences

Lists - a very simple list of values. Can be edited.

Syntax: myList = [1, 2, 3]

Tuples - a variable that holds multiple values, immutable (unchangeable).

Syntax: myTuple = (1, 2, 3)

Range - a range of numbers. The first and last numbers represent the start and stop point. To indicate going down in numeric value, apply a -1.

Syntax: x = range(1, 10) range from 1-10

x = range(10, 1, -1) range from 10-1

Sets - Sets are like lists, but will not show multiple interations of the same value. For example, the data list may be: 1, 2, 3, 3, 3, 4, 5, however the set would read: 1, 2, 3, 4, 5.

Syntax: mySet = {1, 2, 3, 4, 5}

Dictionaries - like a real dictionary, contains mutiple variables each with its own value.

Syntax: postalCodes = {} (creates empty dict)
postalCodes['NY'] = "New York" (adds entry)

Data Types

String - anything in quotes, regular text.
Syntax: "Hello" or single quote 'Hello'

integer - a number with no decimal
Syntax: 1

float - a number with a decimal
Syntax: 1.2345

boolean - only has a value of True or False
Syntax: boolean = True (or) False

complex numbers - math function with imagenary numbers. Ex. 3 + 2i
Syntax: cmplx = complex(3,2)
(to view real part) cmplex.real
(to view complex) cmplex.img

Decimal - more accurate than a float
Syntax: from decimal import * (imports function)
X = Decimal(1)

Fractions - uses fractions
Syntax: from fractions import Fraction

Conditional Programming

If statements - checks to see if a statement is true.
Syntax: if statemnt :
-> execute

Elif - adds another condition if "if" is false

syntax: Elif statement:
execute

Else - runs if "if" is false or elif false

Sytax - else:
execute

Operator Syntax = == equals
< less than
> greater than
<= less than or equal to
>= greater or equal to
!= not equal to

Nested statement - blocks within blocks

How to we list "in" or "not in" (looks in list)

Looping - loops are executed as long as a variable is false, they are terminated when said variable becomes true.

Syntax - done = false
While not done: (not is for false)
execute till true

Operators - += add to current variable
-= subtract
*= multiply
/= divide

break - ends the loop
Syntax: break

Continue - skips a value and continues the loop.
Syntax: continue

For loop - loops ranges, lists, etc.
Syntax: for s in variable:
execute

Can be anything, used for each aspect/element of list, string, dict, etc

Daniel Davis: 304-767-0443
Willis Hall

Functions

~~input = input - non-string~~ # outdated
~~raw = input - string~~ ~~eliminated from python 3~~

Functions

Function are denoted by the () that follows them for arguements.

General Functions

type() - will list data type
int() - changes value to integer if valid
float() - converts integers to floats
print() - prints items

import: some functions are in modules (a file that contains a collection of related functions). These are built-in to python as well as can be downloaded. import brings these functions into a script.
syntax: import()
importing creates what is called a module object. To call up a function from an imported module, use dot notation
Syntax: module.function()

Composition

one of the reasons computers are powerful is that they can take code and compose small bits and pieces into larger programs.

Adding a Function

Functions can be added to python by a user. Doing so is called a function definition

Syntax: def name_function (arguments) *
operations **

* header
** body

The value and type created by def is called a function object. Functions can be nested in eeeh other.

* Flow of Execution

this basically states that all function, variables etc used must be assigned before they can be called on.

** Read a program via its flow of execution, not top to bottom (neccessarily).

Parameters and Arguments

Some functions need or can have arguments. These go in the () of the function. Arguments are assigned to parameters.

i.e

```
def print_name ('Bruce')   <- argument
                ^ Parameter
    print (Bruce)  -> parameter
    print (Bruce)  -> parameter
```

Local

Both vars and parameters are local to a script/Func

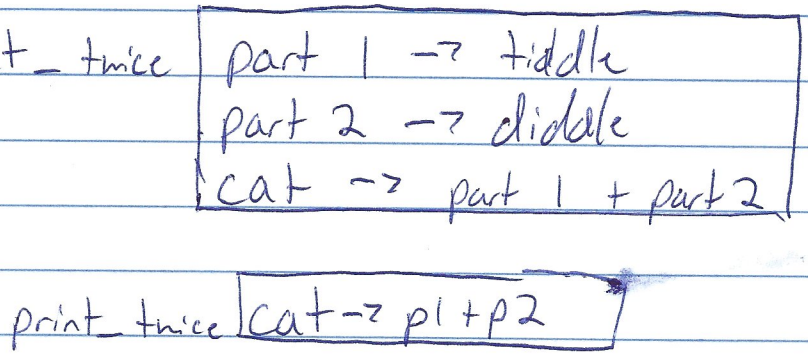
This means that they can only be called upon in that function or that script.

i.e

```
def cat_twice('1', '2')
    cat = '1' + '2'   ← var created in function can only be called upon by that function
    print_twice(cat)
```

* Stack Diagram
  * like a state diagram, but show variables and the function they are in.
  each function is represented by a frame (fig 1)
  A box with the function name

  * fig 1



Frames are arranged in stacks that indicate which function called which. Any variable created outside a function belongs to _main_, which is a name for the top most frame.

If there is an error in a function called, python will give the path of the function and all functions that have called it.

Displaying the location of an error is called a traceback.

Fruitful and Void Functions

When a function yields a result, it is dubbed "fruitful" other functions that only perform an action but yield no return value are dubbed void.

Importing With From

Python has two ways to import modules, direct imports and importing only parts of the module.

Direct import:

Syntax: import module

Parts

Syntax: from module import object

* the diference in the two is that importing directly requires you to call each function in a module using dot notation. If you import each part, the functions can be called up without dot notation.

i.e.	import module # importing module.function # using function	from module import function function
	Direct Import	Parts

** Tip!

you can import all parts of a module by using:
from module import *

Conditionals and Recursion

Modulus Operator

returns the remainder of a division.

syntax: 5 % 3
>>> 2

Boonlean

(For more refer to earlier pgs.)

are either true or false

Logical Operators

There are three; and, or, not. Their semantics are similar to that of English.

* and means both conditions must be true to be true
* or means either or conditions must be true to be true.
* not means a condition must be false to be true.

Conditional Execution.

(refer to earlier notes)

if, elif, else

Recursion

functions can call themselves and or other functions. a function that calls itself is recursive, the process is called recursion.

i.e.

```
def countdown(n):
    if n <= 0:
        print("Blast off!")
    else:
        print(n)
        countdown(n-1)
```

* what happens?

It acts like a loop.

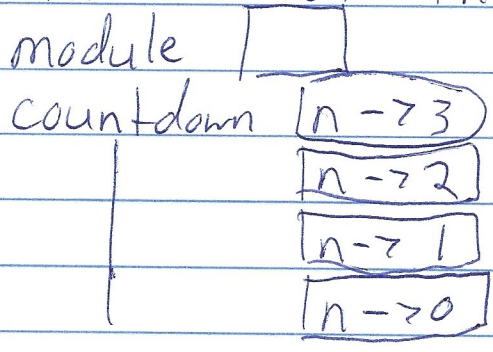
It loops until the function is no longer called, in this case, until the if statement proves true.

Return

return - as far as I know so far, it is used to return a value. They also terminate functions

Stack Diagrams for Recursion

list the interations of the function.



Dead Code - any code following an executed return in a function

Incremental Dereloperment

to avoid long debugging sessions, write small segments of code and debug them. This is called incremental dev.

Composition

Functions can call other functions.
Function can reture booleans.

Checking Types

isistance is a function that checks data types.
syntax: isinstance(Var_here, type_here)

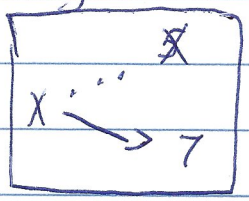
Checks to see if the var matches the data type.

Iteration

Multiple Assignments

You can reasign values to variables at any point in python. Values are not locked.

State diagram for updating vars:



X is no longer 5, it is 7

The while statement

Covered this earlier.

Syntax: while *var *operator *value
loop this code

Break

ends a loops (already covered)

Square Roots

Newtons Method:

$$y = \frac{x + a/x}{2}$$

* this method must be used multiple times to get the correct answer. X is an estimated answer for the square root of a. Keep looping this until the estimant stops changing.

** This is an example of an algorithm.
an algorithm is a mechanical method for solving a category of problems. kinda like an explicit formula.
Algorithms require no intelligence to carry out, they follow a simple set of rules. Designing them does however.

Strings

a string is a sequence. It can be chopped up into pieces.

Indexing

indexing is pulling out a token in a sequence.

i.e

```
fruit = 'banana'
fruit[1]
```

the bracket indicates the index.

indexing starts at zero, so the first letter in a sequence is index [0], the second is [1] and so on.

Len Function

gives the length of a string.

syntax: Var.len

Transversals

processing one character of a string, list, etc. at a time.

i.e

for loops

Searching

example function:

```
def find(word, letter):
    index = 0
    while index < len(word):
        if word[index] == letter
            return index
        index = index +1
    return -1
```

Simple search program. Looks for a letter in a word.

Counting

```
caunt = 0
for a in b:
    if a == c
        count = count +1
print(ccount)
```

Methods

like functions but have a different syntax.

Syntax: word.upper()

uses . (dot notation) to call upon the method, can take arguements.

methods are called invocations. When calling a method it is said to be invoking a method.

Filters

anything the omits anything that is not in agreement with the criteria.

Deleting elements in a list.

pop: a method that removes and saves items ~~from~~ a list if the index is known.
from

Syntax: x = list.pop(index)

del: operator used to remove an item if index is known.

Syntax: del list[index]

remove: a method used to remove an item if element is known.

Syntax: list.remove('element')