4/Jan/18

Date __/__/__
Page _____
STUDY BUDDIES

(4) LOOPS

ways to repeat actions over and over! with less code.

functions: Combine many actions in a single process.

'WHILE' loop

```
while (condition is true):
    do this over & over
```

Ex

```
x = 0
while (x < 10):
    print x
    x += 1
```

- To print 1-9 nos.

```
1 2 3 4 5 6 7 8 9
```

'BREAK' loop

- used to STOP loop

Ex

```
x, y = 0, 0
while (True):
    x += 1
    y += 2
    if (x+y > 10):
        break
```

Date __/__/__
Page _____
STUDY BUDDIES

29001 (A)

'FOR' loops
- for each item in list/range, loop!

```
x = [1, 2, 7]
for i in x:
    print i
```

RANGE
- creates a list of sequential numbers

Ex
```
for i in range(30):
    print i
```
print 0 to 29

Ex
```
for i in range(10, 30, 2):
    print i
```
min, max, step size
from 10 to 30 increment by 2

10, 12, 14 ... 28

CONTINUE (skip)
- start loop over (on next value, if in for loop)

Ex
```
for i in range(30):
    if not (i % 3):
        continue
    print i
```
Non multiple of 3 will be printed.

Date __/__/__
Page _____
STUDY BUDDIES

Exercise / Examples -3

(1)
```
fiboSeq = []
a, b = 0, 1
while (b<1000):
    fiboSeq.append(a)
    a, b = b, a+b
print fibSeq
```
Fibonacci series

(2)
```
factorial = 0        (1)
for i in range(30):  (n)
    factorial += i
print factorial      (*=)
```
factorial of n
if with changes done

(3)
```
primes = []
for i in range(2, 100):
    for x in range(2, i):
        if (i % x == 0):
            break
        else:
            primes.append(i)
print primes
```

Date __/__/__
Page
STUDY BUDDIES

(5) Exception Handling

- Overview
- try / except
- pass
- raise
- finally
- examples

Overview

- Prevents codes and scripts from Breaking!
- Can be used for handling user inputs

TRY / EXCEPT

TRY

- 'try' TO EXECUTE the code below...
- may be used anywhere that keyboard user input is required

Except

- CATCHES all ERRORS! or can just catch a specific error
- May be used anywhere that keyboard user input is required.

Ex **Break the code!!** 😊

```
x = 5 + 'raj'
```

Date __/__/__
Page _____
STUDY BUDDIES

Handling

```
try:
    x = 5 + 'raj'
except:
    print 'damn it'
```

→ correct it / patch it

PASS

- says to IGNORE and move on
- may be used in for, while, Try/Except instances

Ex

```
try:
    x = 5 + 'raj'
except:
    pass
```

RAISE

- ~~Force~~ force an error to occur

Ex `raise TypeError("hahaha")`

#

```
Traceback (most recent call last):
  File --
TypeError: hahaha
```

FINALLY

- Last actions to perform following 'try' & 'except'
- Occurs before any real errors are returned.

Ex

in
```
try:
    x = 5 + 'ray'
except ZeroDivisionError:
    print 'will not see this'
finally:
    print 'the final word'
```

op
```
the final word
Traceback (most ...
```
→ it comes first then Real Error comes

Examples

①
```
for i in [[1,2,3],[4,5],[6,7]]:
    for j in i:
        if (4<j<=6):
            print j
```

②
```
for i in range(10):
    if (i%2):
        pass
    else:
        continue
print i
```

③
```
try:
    x = 1.0/0.0
except ZeroDivisionError:
    print "got it!"
finally:
    raise TypeError("Just kidding!")
```

6/Jan/ff

Date __/__/__
Page _____
STUDY BUDDIES

(6) FUNCTIONS

- Way to combine many actions into a single process, for reuse later.

Ex Wash Dish = LOOP

Wash dishes
sweep
mop
degnome
} := clean kitchen (FUNCTION)

syntax

define or declare a function

INPUTS arguments passed in here

```
>>> def doesNothing():
        pass
            ↖ Your main code here

>>> doesNothing()
        ↖ call your function
```

RETURN

- a way to OUTPUT or return data from a fn.

Ex

```
def makeOne():
    return 1
x = makeOne()
print x      → 1
```

Date __/__/__
Page _____
STUDY BUDDIES

Argument Types

Regular Argument → var1
keyword argument → var2 = 3

```
def myFunc (var1, var2 = 3):
    ...
```

↑ Keyword args set DEFAULT value that MAY be overridden

Ex
```
def addTen (myInt):
    myInt += 10
    return myInt

x = 12
dir()
y = addTen(x)
print x, y
```

dir() → display which are created & stored in current file.

```
12, 22
```

LOCAL VS GLOBAL VARIABLES

LOCAL: variables created and stored within a fn. that will be deleted from memory when the fn. completes

Ex
```
def myFunc():
    localVar = 5
```

GLOBAL: variable that accessable anywhere within program.
- Uses keyword 'global'

Ex
```
glVar = 5
def myFunc():
    global glVar
```

Date __/__/__
Page _____
STUDY BUDDIES

Documenting & comments

- Not necessary for code to run.
- Primarily used for debugging, and breaking down code for other programm

~~Documenting~~

""" my description """

Document string

- Text describing the function
- Comes immediately after function creation
- use triple quotes to enclose.

Ex

```
def myFunc():
    """my description"""
```

or """ ___ """

Comments

```
# comment 1
x = 5  # 2
# 3
```

Ex

```
def func()
    '''
    This is documented by me
    '''
    # Do nothing
    pass

print func.__doc__
```

This is document by Me

Date / /
Page
STUDY BUDDIES

Examples - 4

(1)
```
def returnTwo()
    return 20,30
x,y = returnTwo()
print x, y
```

x y
20,30

(2)
```
def mul(x,y):
    return x*y
## FACTORIAL
print reduce(mul, range(1,11))
```

3628800

(3)
```
def cubeFn(x):
    return x*x*x
print map(cubefn, range(1,11))
```

1, 8, 27, 64, 125, 216, 343, 512, 729, 1000

(4)
```
def myAdd(var1, var2 = 10):
    return var1 + var2
print myAdd(7)      → 17
print myAdd(8,5)    → 13
```

8/Jan/17

Date __/__/__
Page _____
STUDY BUDDIES

(7) Files and User Input

- Files that ends in '.py', '.pyw', '.pyc'
  - .pyc = Python compile file (Only computer understands)
  - .py = Python writeable file
  - .pyw = —— " ——

Benifits of FILE
- Flexible, Editable
- Save for reuse in future
- No more Retyping
- Run whenever ready

RAW_INPUT / INPUT
- way of getting keyboard user input.

s = raw_input("Enter any no: ")

s: Store user input within var. | raw_input: Function

OR s = input("Enter any no: ")

IMPORTING
- way access fn., classes & variables from other files.

Ex import random

import: keyword | random: lib or file

Date __/__/__
Page _____
STUDY BUDDIES

Examples

GuessingGame.py (without importing any lib)

```
highest = 10
ans = 7
guess = input ("Guess a number from 0 to %d:"
        % highest)
while (int(guess) != ans):
    if (int(guess) < ans):
        print "Answer is higher"
    else:
        print "Answer is lower"
    guess = input("Guess a number from 0 to %d:" %highest)
print "You WIN"
```

~~GuessingGame.py~~ (Using ~~import random~~)

~~highest = 100~~

~~ans~~

GussingGame.py (Using import)

```
import random
highest = 10
ans = random.randrange (highest)
guess = ...
  ...
```

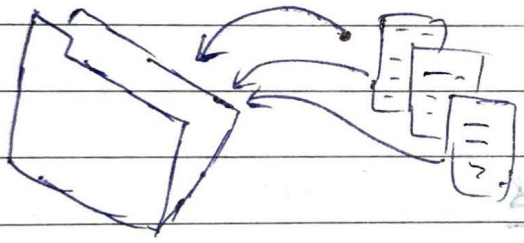
Date __/__/__
Page _____
STUDY BUDDIES

SEPERATE FILES

- break up code into modules, or chunks based on functionality.
- Prevents having super long files & having to scroll forever.
- MUCH EASIER to manage
- Allows for easier reuse in the future

How?

- Must ~~ben~~ be in same folder or directory



Ex

hello.py

```
def printHello():
    print("Hello")
```

main.py

```
import hello

for i in range(5):
    hello.printHello()
raw_input("End of line")
```

folder is saved

Ex

O/P

```
Hello
Hello
Hello
Hello
Hello
End of line
```

Date __/__/__
Page
STUDY BUDDIES

Mod Ex.

(1)

f1.py

```
def test1():
    pass
def printHello():
    print "Hello"
```

main.py

```
print dir()
from f1 import *
print dir()
printHello()
raw_input("DONE")
```

(2)

f1.py

```
def test1():
    print 'testing'
def printHello():
    print 'Hello'
```

main.py

```
import f1 as f
f.test1()
f.printHello()
```

(8) Classes

- class
- self
- initialization
- calling fn.
- create a hero
- Examples

Def.* Way of packaging variables and functions together

Begin with 'class keyword' → `class Test:` ← ALWAYS capitalize 1st letter of class name

Ex (1)

```
    pass
x = Test()
```

← create class instance is just like calling a function.

Date __/__/__
Page
STUDY BUDDIES

Calling Functions

Ex

```
class Ph:
    def __init__(self):
        self.y = 5
        self.printHello()
    def printHello(self):
        print 'Hello'
x = Ph()
```

variable Name

Hello

Ex Creating a Hero

```
class Hero:
    def __init__(self, name):
        self.name = name
        self.health = 100

    def eat(self, food):
        if (food == 'apple'):
            self.health -= 100
        elif (food == 'banana'):
            self.health += 20
Bob = Hero("Bob")
print Bob.name
print Bob.health
Bob.eat('apple')
print Bob.health
```

var. names

This comes by default in every functions of in the class.

Bob
100
0

banana

Bob
100
120

Date __/__/__
Page _____
STUDY BUDDIES

Examples - 5

(1)
```
class Hello:
    def __init__(self):
        self.name = 'Rajiv'
        self.healthBonus = 10
class Hero:
    def __init__(self):
        self.health = 100
    def eat(self, food):
        self.health += food.healthBonus
bob = Hero()
R = Hello()
bob.eat(R)
print bob.health
```

110

(2)
```
class AddFive:
    def add(self, var):
        return var + 5
class Num:
    def __init__(self, value):
        self.val = value
        self.OtherVar = None
n = Num(15)
a5 = AddFive()
n.OtherVal = a5.add(n.val)
print n.val, n.OtherVal
```