Types

str	'abcd'
int	1,2,3,4
float	1.23
bool	True/False
None	None
type('abc')	shows the type of the object
type conversion	int('123') --> 123
bool(x)	False for x values that are null or empty x. True for otherwise
Naming conven-tions	lowercase and underscore (_HAPPYday, happyday)

Order of growth

def factorial(n): if n < 2: return 1 return n * factorial(n - 1)	O(n)
def fib(n): if n < 2: return 1 return fib(n - 1) + fib(n - 2)	O(2^n) (each branch produces another 2 branches)
def foo(n): while n > 1: print(n) n = n // 2	O(log n)
def bar(n): while n > 1: x = n while x > 1: print(n, x) x -= 1 n -= 1	O(n^2) (loop in a loop)

Order of growth (cont)

def baz(n): while n > 1: x = n while x > 1: print(n, x) x = x // 2 n -= 1	O(n log n) (log n in a loop)
String and tuple slicing	Slicing is an O(n) operation as you **need to create another string/tuple when you slice**.
def count_me(n): if n < 10: return n else: return count_me(n + 1)	O(infinity)
String and tuple concatenation	O(n)

Time complexity--> looking at the branches of the tree
Space complexity--> looking at the depth of the tree (depends on the pending operations the function incurred)

And also for analysing OOG questions:
for <loop>:
<do work>
Time
= work(1) + work(2)+...+work(n)
If work is fixed (let's say work = k) regardless of n, Time = n * k
Otherwise,
Time = sum of work
Remember to state what the n means

Recursive and Iteration

Recursion

```
def recusi ve_ fn(n):
        if <base case>:
                return <base case value>
        else:
```

Recursive and Iteration (cont)

```
>        return <manipulate recursive_fn(n-1) such that it becomes nth case>
```

Iteration (for or while loop)

```
def iterative_fn(n):
    temp = <some basic value>
    for i in <range>:
        <update temp>
    return temp
def factorial (n):
    product , counter = 1,1
    while counter <= n:
        product *= counter
        counter += 1
    return product
```

Tuples comparison

Tuples are compared **position by position:** the **first item of the first tuple is compared to the first item of the second tuple** (same as string); if they are not equal (i.e. the first is greater or smaller than the second) then that's the result of the comparison, else the second item is considered, then the third and so on.

Truth Tables

OR	True	False
True	True	True
False	True	False

AND	True	False
True	True	False
False	False	False

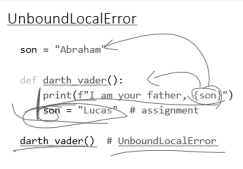
NOT	
True	False
False	True

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Truth values revisited

Truth and False will be represented by 1 and 0 respectively.
not 'abc' --> False
1 or 0 --> 0
'' is same as False is same as 0 is same as []
while True: infinite looping
while False: will not loop at all

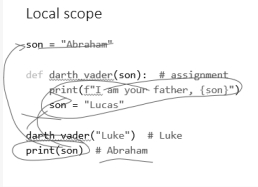
UnboundLocalError



String and tuple slicing

s[start:stop(**noninclusive**):step]
eg) s = " 123 45", d ='6789'
concat ena te- -> s + d--> '12345 6789'
s[0] = " 1", s[1:]= " 234 5", s[:2]= " 12", s[1:5:2] = " 13"
s[-1] = " 5", s[-1:-3] = " ", s[-3:-1]= '34', s[-5:- 1:2]= '13', s[-5:- 1:-2]= ''', s[-1:- 5:-2] = '53'

Variable (Global and Local)



Variable is local if assigned to in function
Namespaces match function and variable names to their definitions and assignments

Random module

random.randint(a, b)	Return a random integer N such that a <= N <= b.
random.randrange(start, stop[, step])	Return a randomly selected element from range(start, stop, step). This is equivalent to choice(range(start, stop, step)), but doesn't actually build a range object
random.random()	Return the next random floating point number in the range [0.0, 1.0).
random.random() can be scaled	int(random.random() * 9999) → 3658

break

```
for val in "string":
        if val == " i":
                break
         pri nt(val)
print( "The end")
```

Output:

```
s
t
r
The end
```

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Hiding 'something'

Input an empty **(lambda : "Prof is really cool")**
In that way what is printed or sliced /ac cessed will be the function itself.
To recover the secret message, can input () at the back.

Functions

```
def <name>(formal parameters):
      body
    return (something if not it is going to return to zero)
```

Anonymous function→ **lambda**
lambda x: x+ 1
A function need not use every parameter passed into it. (def foo(a, b): return a + a)
A function without any parameters or return value is still valid. (def foo(): return)
Functions do not necess arily need a return value.(def foo(a, b, c):a + b + c)

Conditionals

```
if <statement>:
        do something
elif <st ate men t>:
        do something
else <st ate men t>:
      do something
while <st ate men t>:
      do something
```

Equality/ in function

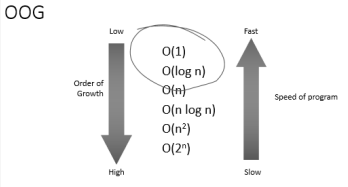
is	This means the same object. (3 is 3)--> True
==	This mean the same content. (3.00 == 3) --> True

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Equality/ in function (cont)

in	check the elements inside the other group. Eg) x = (1,2,3). 1 in x--> True

OOG Time and Speed



Different Types of Error

NameError	Raised when a variable is not found in local or global scope.
IndexError	Raised when index of a sequence is out of range.
RuntimeError	Raised when an error does not fall under any other category.
SyntaxError	Raised by parser when syntax error is encountered
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable
RecursionError	Infinite looping

Tuple and string functions

len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sum()	Retrun the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.
count()	count the occurrences of an element in a tuple
index()	Searches the string for a specified value and returns the position of where it was found

continue

```
for val in "string":
       if val == " i":
                con tinue
        pri nt(val)
print( "The end")
Output:
s
t
r
n
g
The end
```

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com