Sample Function:

```python
def isPerfect(number):
    sum = 0
    factor = 1
    while factor < number:
        if number % factor == 0:
            sum += factor
        factor += 1
    return sum == number
```

Types:

```python
>>> type(13.2)
<type 'float'>
>>> type('abc')
<type 'str'>
>>> x = 2
>>> type(x)
<type 'int'>
>>> isinstance(x, int)
True
>>> isinstance(x, float)
False
>>> s = "12.34"
>>> x = float(s)
>>> x
12.34
>>> y = int(s)
...error message...
>>> s = str(3.14159)
>>> s
'3.14159'
```

Strings & Lists:

```python
>>> word = "ABCDEFG"
>>> len(word)
7
>>> word[2]
'C'
>>> word[1:3]
'BC'
>>> word[3:]
'DEFG'
>>> word[:3]
'ABC'
>>> 'G' in word
True
>>> 'H' in word
False
>>> word + "end"
'ABCDEFGend'
>>> word.index("B")
1
>>> word[-2]
'F'
```

Dictionaries:

```python
>>> marks={'y':96,'x':57,'z':82}
>>> len(marks)
3
>>> marks['y']
96
>>> marks.get('y',42)
96
>>> marks.get('w',42)
42
>>> 'x' in marks
True
>>> 'w' in marks
False
>>> 57 in marks
False
>>> for key in marks:
...    print key
...    print marks[key]
    print
y
96
x
57
z
```

```python
>>> numbers=[3,1,4,1,5,9]
>>> len(numbers)
6
>>> numbers[2]
4
>>> numbers[1:3]
[1, 4]
>>> numbers[4:]
[5, 9]
>>> 4 in numbers
True
>>> del numbers[4]
>>> numbers
[3, 1, 4, 1, 9]
>>> numbers.append(42)
>>> numbers
[3, 1, 4, 1, 9, 42]
>>> numbers.extend([5,6])
>>> numbers
[3, 1, 4, 1, 9, 42, 5, 6]
>>> numbers[2:5]=[7,8]
>>> numbers
[3, 1, 7, 8, 42, 5, 6]
```
Format Examples:

```python
>>> "A={},B={}".format(15-4,42+3)
'A=11,B=45'
>>> "n is {:8d}".format(426)
'n is      426'
>>> "n is {:12.4f}".format(3.14159265)
'n is       3.1416'
>>> "n is {:0.4f}".format(3.14159265)
'n is 3.1416'
```

Linked List Reminders:
- Each node is a dict with two entries.
- The 'data' entry holds the data in the node.
- The 'next' entry is the next node in the list (technically, the address of the next node).
- The last node's 'next' entry is None.
- We refer to a list by referring to the first node (technically, the address of the first node).

Binary Search Tree Reminders:
- Each node is a dict with three entries.
- The 'data' entry stores the data in the node.
- The 'left' and 'right' entries are the left and right children of the node (technically, the addresses of the children).
- If a node has no left child, its 'left' entry is None.
- Same for no right child.
- We refer to a tree by referring to the root (topmost) node.
- Every value in the subtree to the left of a node must be equal to or less than the value in the node.
- Every value in the subtree to the right of a node must be equal to or greater than the value in the node.

Summation Formulas Used For Complexity:

\[
\sum_{i=1}^{n-1} i^p = O\left(n^{p+1}\right)
\]

(same complexity if i goes from 0 to n)

\[
\sum_{i=0}^{n} b^i = \frac{b^{n+1} - 1}{b - 1}
\]