Other loop controls - break, continue, else

- In some cases we may want to break out in the middle of a loop on a condition. The `break` command can be used for this.
- Some times we want to avoid executing a part of the loop body in such cases the `continue` command is useful.
- Loops can have an else clause. It is executed when the loop is exited normally (i.e. without using break) via the conditional expression during the continue loop test.

Program to check if a positive integer > 1 is prime or not.

Other loop controls - break, continue, else

- In some cases we may want to break out in the middle of a loop on a condition. The `break` command can be used for this.
- Some times we want to avoid executing a part of the loop body in such cases the `continue` command is useful.
- Loops can have an else clause. It is executed when the loop is exited normally (i.e. without using break) via the conditional expression during the continue loop test.

Program to check if a positive integer > 1 is prime or not.

```
n=int(input('Enter a positive integer='))
if n<=1:
    ans='Illegal input'
else:
    for i in range(2,n//2+1):
        if n%i==0:
            ans='Not prime'
            break
    else:
        ans='Is prime'
print(ans)
```

Functions

- Complex programs require code reuse since the same or similar computations are required to be done at multiple places. So, this requires ecapsulating certain kinds of behaviour in a reusable abstraction.
- A function is one such widely used abstraction. It can be reused by calling it with different arguments at different times.
- The simplest and most commond definition is:

  ```
  def FnName(arg-seq):
      FnBody
  ```

  A function can be called by `FnName(actual-args)`.
- Function definitions can be nested.

More on function arguments

- We can specify default values for some arguments or all arguments.

```
def argType(arg1='first', arg2='second', arg3='third'):
    print('arg1=',arg1)
    print('arg2=',arg2)
    print('arg3=',arg3)
```

  All non-default arguments must occur before the first default one. So, an argument sequence like, `(arg1='first', arg2, arg3='third')` is wrong.

- When the function is invoked all non-default arguments must be given. Default arguments are optional. The function will use the default values if the argument is not given. Arguments can be given using argName=argValue in any order. For example, `argType(arg3='one', arg2='two', arg1='three')`

Anonymous/lambda functions

Functions without a name can be defined and associated with a variable as a value. They are often used when temporary functions are needed for example in filter and map.

```
cube=lambda x: x*x*x

odd=list(filter(lambda x: x%2!=0, [3,9,8,2,6]))

sq=list(map(lambda x: x*x, [3,9,7,2]))
```

Functions are first class objects

- Functions are first class objects - that is they can be returned as values and passed as arguments.

```
def f(x):
    return 2*x

def g(y,z):
    return y(z)

h=f
```

Recursion

- A recursive function is one that is defined in terms of itself. They always have some base cases (non-recursive) and some recursive cases.
- Recursive definitions can be very elegant but can be computationally expensive if applied without thought.
- Examples of recursive definitions:

```
f(0)=f(1)=1, f(n)=n*f(n-1) - factorial function

f(1)=f(2)=1, f(n)=f(n-1)+f(n-2) - Fibonacci function

Exponentiation (i.e. n^m by recursion)
f(n,0)=1, f(n,1)=n
f(n,m)= if (even(m)) then f(n,m//2)**2 else n*f(n,m-1)
```

Names and binding

Python does not declare names/vars. Names are bound to values when::

- They are assigned within a function.
- They are arguments of a function and function invocation takes place.
- They are in for, import statements.
- Whenever names are assigned in a function they become local to the function.
- Names can be used only after they are assigned otherwise there is an error.

Scope rules

Definition 1 (Scope)

The scope of a variable or name is the program text in which a particular binding (that is the object/value to which it is bound) for that variable is active or visible. ◁

When a variable is by itself (i.e. without a dot '.') the LEGB rules apply for name resolution:

- Local: Arguments of def or lambda functions, variables assigned in the body of a function or lambda are local to that function body provided they are not declared global or nonlocal.
- Enclosing: variables assigned in enclosing function - that is def inside def.
- Global: variables assigned at the top level or those declared global inside def/lambda.
- Built-in: variables assigned in the built-in names module.