# PYTHON – WHILE AND FOR LOOPS

# While Loop Basic Format

```
while <test>:          # Loop test
  <statements1>        # Loop body
else:                  # Optional else
  <statements2>        # Run if didn't exit
  loop with break
```

# EXAMPLE

```
while True:
 print('Can't stop me!')

x = 'spam'
while x:
 print(x, end=' ')
 x = x[1:]

???
```

# EXAMPLE

```
a=0; b=10
while a < b:
 print(a, end=' ')
 a += 1


while True:
 loop body...
 if exitTest(): break
```

# LOOP STOPERS

- break
  - Jumps out of the closest enclosing loop (past the entire loop statement)
- continue
  - Jumps to the top of the closest enclosing loop (to the loop's header line)
- pass
  - Does nothing at all: it's an empty statement placeholder

- Loop else block
  - Runs if and only if the loop is exited normally (i.e., without hitting a break)

# General loop format

```
a=0; b=10
while a<b:
 if a== 5: continue
 a+=1
 if a== 6: break
else:
 print('EXIT')
```

What is the output?

# Example (continue)

```
x=20
while x:
 x -= 1
 if x % 2 != 0: continue
 print(x, end=' ')
```

# Example (`break`)

```
y=input("enter a number to be checked for
 factor")
x = y // 2
 while x > 1:
     if y % x == 0:
          print(y, 'has factor', x)
          break # Skip else
     x -= 1
 else:
     print(y, 'is prime')
```

# `pass` statement

Kill the CPU! -> while True: pass

As a place holder ->

```
def func1():
    pass
def func2():
    pass
```

FYI ellipses also work as place holder..

```
def func1():
    ...
def func2():
    ...
```

# For Loops: good for iterating sequences

```
sum = 0; l=[1,2,3,4]
for x in l:
 sum = sum + x

l=[4,5,6,7]; prod = 1
for item in range(len(l)):
 prod *= l[item]
```

# Looping Dictionaries

```
list(D.items())
OUTPUT:
     [('a', 1), ('c', 3), ('b', 2)]

for (key, value) in D.items():
     print(key, '=>', value)
OUTPUT:
     a => 1
     c => 3
     b => 2
```

# `in` operator

```
items = ["aaa", 111, (4, 5), 2.01]
tests = [(4, 5), 3.14]
for key in tests:
 if key in items:
     print(key, "was found")
 else:
     print(key, "not found!")
```

# Range Function

- Produces an iteratable object (range) of integers
- A general tool, can be used in a variety of contexts.
- Most often to generate indexes in “for”
- Wrap it in a list call to display its results all at once

```
>>> list(range(5)), list(range(2, 5)),
 list(range(0, 10, 2))
([0, 1, 2, 3, 4], [2, 3, 4], [0, 2, 4, 6, 8])
>>> list(range(-5, 5)), list(range(5, -5, -1))
([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4], [5, 4, 3,
 2, 1, 0, -1, -2, -3, -4])
```