L#8. Introduction to python: programming structures nested-loops

Ago 2017

Multiple Assignment

FILE: multipleAssignment01.py

We can assign values to more than one variable using just a single line

assignment	print(x, y, z)
x,y,z=1,2,3	1 2 3
x,y,z=1,2,'pepe'	1 2 pepe # variables can be of different types
x,y,z=1,2	error
x, y,*z=1,2,3,4,5	1 2 [3, 4, 5] # *z stores remaining values in a list [3, 4, 5]
x=1; y=2; z=3	1 2 3
x=y=z=3	3 3 3
x=y=z='pepe'	pepe pepe pepe
x,y,z='pepe',[1,2,3],(4,5,6)	pepe [1, 2, 3] (4, 5, 6) # variables can be of different types

Assignment Operators

Operator	Example	Equals To
=	a = 10	a = 10
+=	a += 10	a = a+10
-=	a -= 10	a = a-10
*=	a *= 10	a = a*10
/=	a /= 10	a = a / 10
%=	a %= 10	a = a % 10
//=	a //= 10	a = a // 10
**=	a **= 10	a = a ** 10
&=	a &= 10	a = a & 10
|=	a |= 10	a = a |10
^=	a ^= 10	a = a ^10
>>=	a >>= 10	a = a >> 10
<<=	a <<= 10	a = a << 10

Examples: Rounding in Python

		21.324	21.5	21.689	-21.324
round(x)	nearest int	21	22	22	-21
math.ceiling(x)	=> +∞	22	22	22	-21
math.floor(x)	-∞ <=	21	21	21	-22
math.trunc(x)	chop decimals	21	21	21	-21
numpy.round	for arrays	21.0	21.0	22.0	-21.0

FILE: rounding.py

break and continue statements

break and continue can be used to interrupt the normal operation of a while or for loop

break terminates the execution of the current cycle (nearest enclosing loop) and stops the loop and passes control to the next statement after the loop

continue statement interrupts the current cycle and returns control to the next cycle; loops continues until the end.

Both break and continue must be within a loop, otherwise an error message is obtained

break, example-1

```
for i in range(1,6,1):
    if i ==3:
        break

    print('i= ', i)

print('After loop')
print('Have a nice day')
```

1,2,3,4,5

Output:

i= 1

i= 2

After loop

Have a nice day

Break statement terminates the execution of the loop and passes control to the next statement after the loop

break, example-2

PSEUDOCODE:

```
SET flag =1, k=2
INPUT N
WHILE k<=(N-1)
  IF N % k ==0
    flag=0

  k=k+1


IF flag==1
  PRINT N & " is prime"
ELSE
  PRINT N & "is not prime"
```

Improved in Python

```
# break statement in Python

flag=1; k=2
N = int(input("Enter number to check if prime: "))
while k<=N-1:
    if N % k ==0:
        flag=0
        break          # stops loop
    k=k+1

if flag==1:
    print(N, " is prime")
else:
    print(N," is not prime")
# file: prime1.py
```

Rewrite the code using this improvement:

```
flag=0
print(N," is not prime")
break
```

```
if flag==1:
    print(N, " is prime")
```

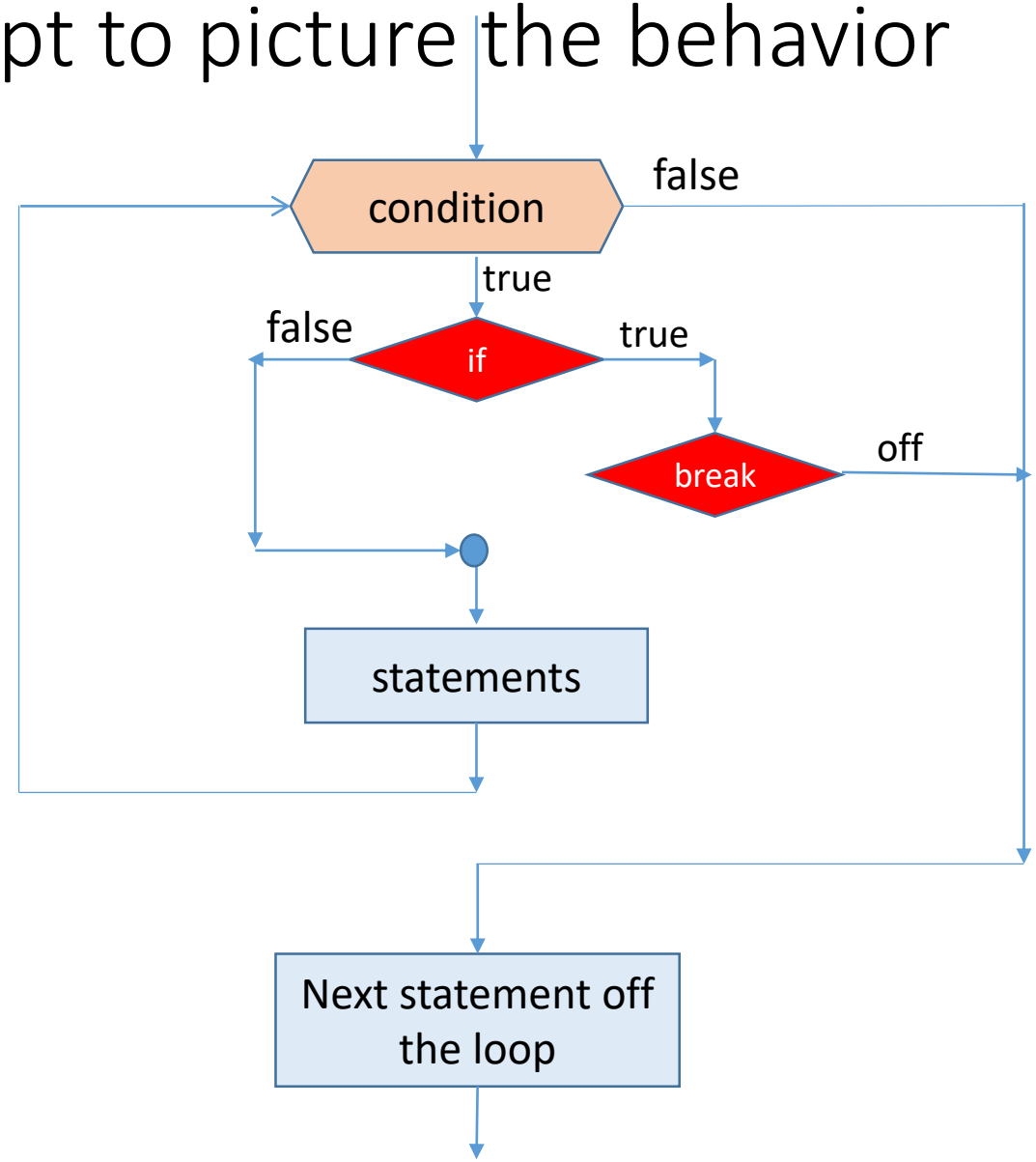
OUTPUT

```
Enter number to check if prime: 13
13  is prime

Enter number to check if prime: 15
15 is not prime
```

Recall from L#6

This algorithm finds primes and it's called Trial Division

break: an attempt to picture the behavior



break statement stops the loop

continue, example-1


```
for val in range(1,6,1):
    if val==3:
        continue

    print('val=  %d ' %(val))

print('After loop')
```

```
for val in range(1,6,1):
    if val==3: continue
    print('val=  %d ' %(val))
print('After loop')
```

Output:

val= 1
val= 2
val= 4
val= 5
After loop

Note that val==3 is missing on output. The *continue* statement terminates the current pass through the loop and returns control to the top of the loop

continue example-2

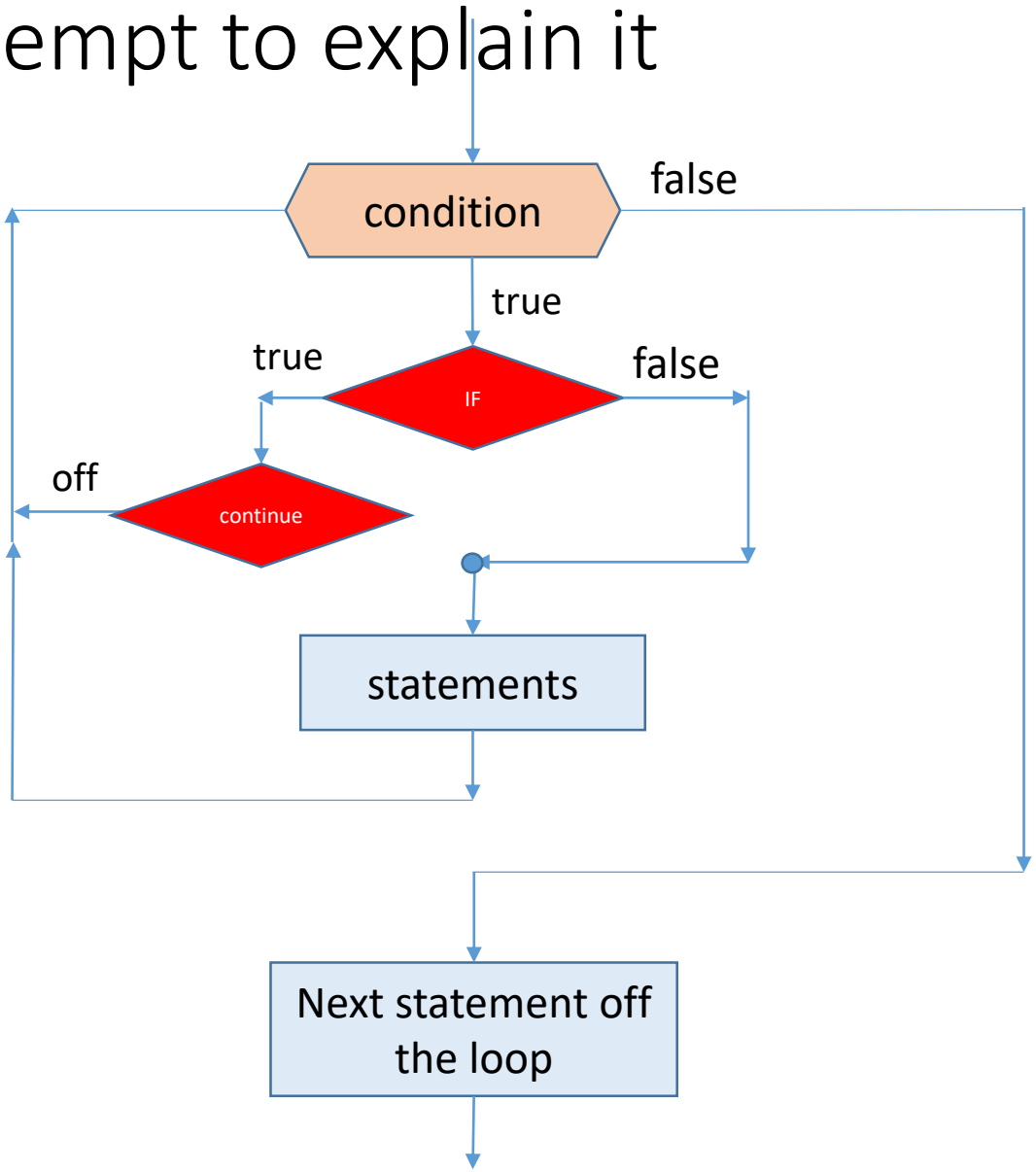
- The continue statement terminates the current iteration and continues with the next:

```
for n in range(1,11,1):
    if n % 2 ==0:
        print('Even number:',n)
        continue

    print('Not even number:',n)
```

OUTPUT

```
Not even number: 1
Even number: 2
Not even number: 3
Even number: 4
Not even number: 5
Even number: 6
Not even number: 7
Even number: 8
Not even number: 9
Even number: 10
```

Continue: an attempt to explain it



else clause in loops

- Loop statements may have an *else* clause. It is executed when the loop ends the list (with for) or when the condition becomes false (with while), but not when the loop is ended by a break statement.
- Example:

```
while condition:
        statement_1
        ...
        statement_n
else:   # condition=False
        statement_1
        ...
        statement_n
```

```
for item in sequence:
        statement_1
        ...
        statement_n
else:  # after last item in sequence
        statement_1
        ...
        statement_n
```

- This is exemplified by the following two slides, which searches for prime numbers:

Is N prime?

Code

```
# introduces the else clause
k=2
N = int(input("Enter a number>1 to check if prime: "))

while k<=N-1:
    if N % k ==0:
        print(N," is not prime")
        break
    k=k+1
else:
    print(N," is prime")

print("Have a nice day")
```

Output

```
Run-1:
Enter a number>1 to check if prime: 13
13 is prime
Have a nice day


Run-2:
Enter a number>1 to check if prime: 14
14 is not prime
Have a nice day
```

QUIZ: Use for instead of while

are numbers in a sequence **prime**?

Are numbers in [2,3,4,5,6,7,8,9] prime?

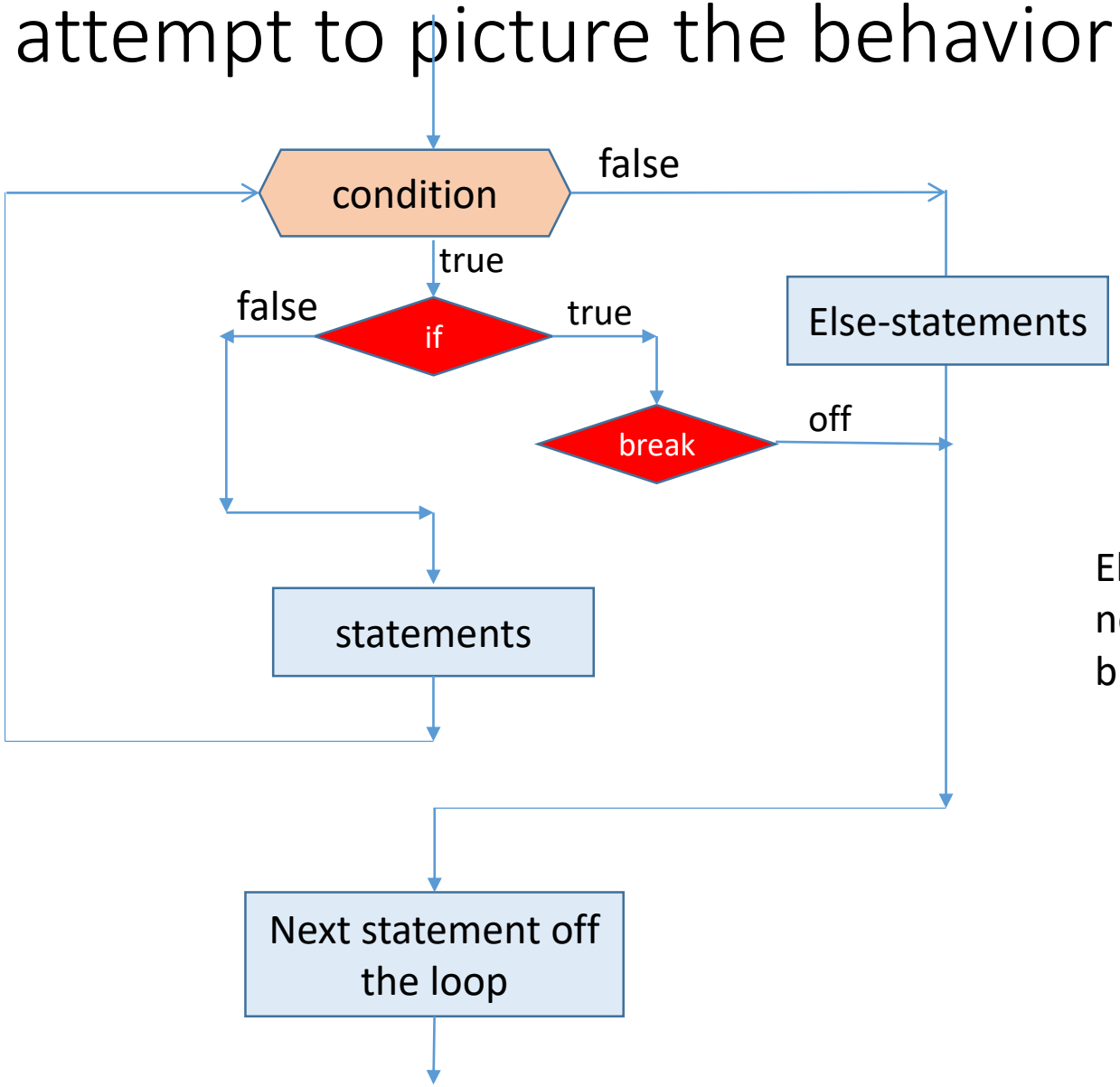
```
# introduces the else clause

for N in range(2,10):
    for k in range(2,N):
        if N % k ==0:
            print(N," is not prime")
            break
    else:
        print(N," is  prime")

print("Have a nice day")
```

Output

```
2  is  prime
3  is  prime
4  is not prime
5  is  prime
6  is not prime
7  is  prime
8  is not prime
9  is not prime
Have a nice day
```

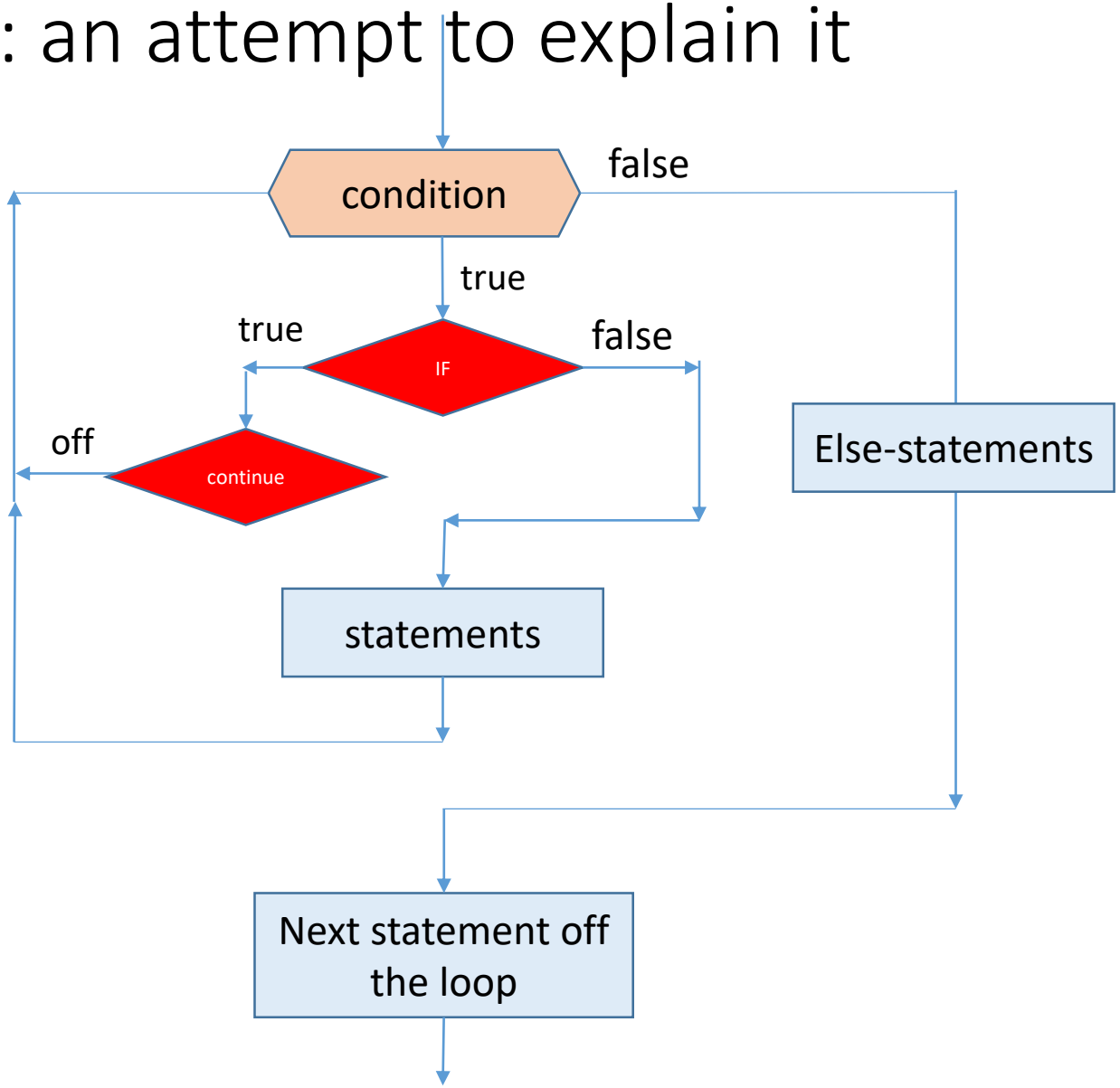
Here we have 2 nested loops: one “inner” and one “outer”
“else” clause belongs to the inner loop (pay attention to indentation)

break & else: an attempt to picture the behavior



Else statements are not executed if break is executed

Continue & else: an attempt to explain it



Code clarity

$$area = \pi r^2$$

Five steps:

```
import math
R = input(“Enter radius”)
r = float(R)
A=math.pi*r**2
print(“The area is “, A)
```

Three steps:

```
import math
r = float(input(“Enter radius”)
print(“The area is “, math.pi*r**2)
```

Two step:

```
import math
print("The area is ",math.pi*float(input("Enter radious "))**2)
```

Reference: How to Think Like a Computer Scientist: Learning with Python 3 Documentation by Wentworth

Breaking long lines in Python

Use parenthesis

Original:

```
s = 'Area: {0}, Estimated ({1}): {2}'.format(area_of_circle, points, estimate(radius, points))
```

Splitted:

```
s = ('Area: {0}, Estimated ({1}): {2}'
    .format(area_of_circle, points, estimate(radius, points)))
```

Note the extra beginning and the ending parenthesis.

Original:

```
s1 = x + x**2/2 + x**3/3 + x**4/4 + x**5/5 + x**6/6 + x**7/7 + x**8/8
```

Splitted:

```
s3 = (x + x**2/2 + x**3/3
    + x**4/4 + x**5/5
    + x**6/6 + x**7/7
    + x**8/8)
```

https://doingmathwithpython.github.io/breaking-long-lines-in-python.html

Breaking long lines in Python

Use the line continuation operator

```
s3 = x + x**2/2 + x**3/3   \
     + x**4/4 + x**5/5   \
     + x**6/6 + x**7/7   \
     + x**8/8
```

When calling functions

Press enter and without doing anything more keep writing your statement over multiple lines. For example:

```
x = 1
print(x,
      x)
print('Area: {0}, Estimated ({1}): {2}'.format(area_of_circle,
                                               points,
                                               estimate(radius, points)))
```

Using parenthesis

```
if (cond1 and cond2 and cond3
    and cond4):
    # True block
else:
    # False block
```

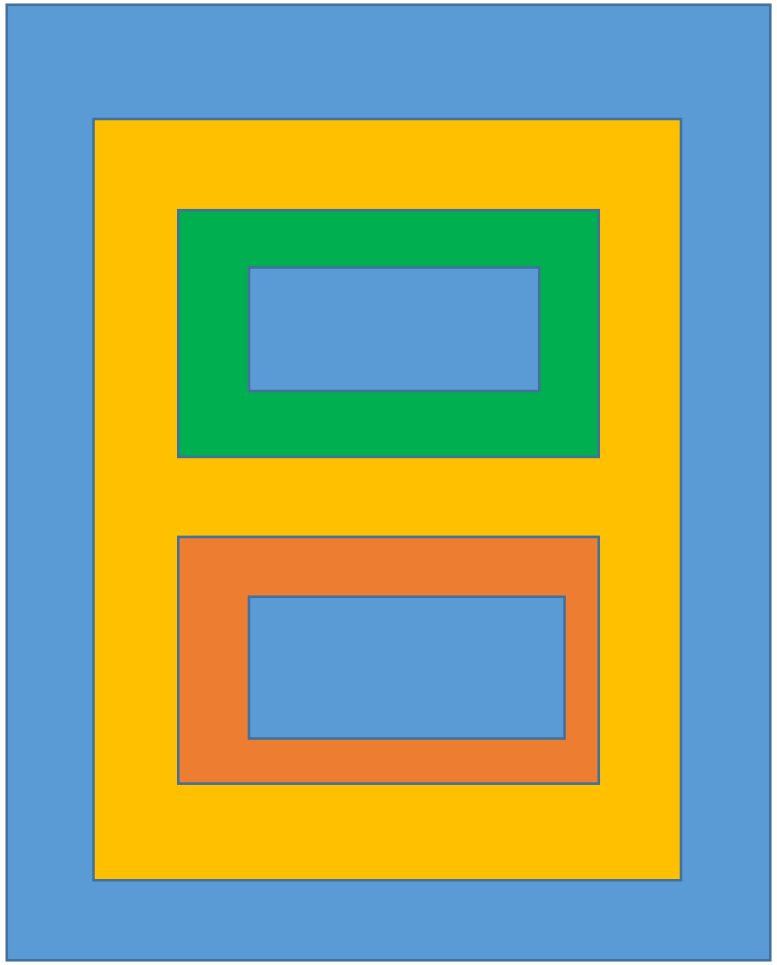
Using line continuation operator

```
if cond1 and cond2 and cond3 \
   and cond4:
    # True block
else:
    # False block
```

https://doingmathwithpython.github.io/breaking-long-lines-in-python.html

Nested Loops

- Python statements can go inside the body of structures, such as, if, loops, etc.
- A *for* loop can go inside the body of another *for* loop (nested). A *while* loop can go inside another *while* loop. Combinations of *for* loop and *while* loop are also possible
- No restrictions for the number of nested loops
- For two nested loops, the inner loop executes completely before the outer loop's next iteration.
- ~~Read "Nesting Loops Article.docx" and "002-Loops Exercises.pdf"~~



Nested Loops Top-down design: Print Triangle

Print triangle of numbers:

<u>Output:</u>
12345
2345
345
45
5

Simpler

<u>Output:</u>
12345
12345
12345
12345
12345

Simpler

<u>Output:</u>
12345

KEY: Creating a simpler problem out of a complex one, then constructing the complex one by modifying the simpler one(s)

The simplest:

```
for n in range(1,6,1):
    print(n)

for n in range(1,6,1):
    print(n,end='')
```

Output:

```
1
2
3
4
5


12345
```

Complex Problem Solving Beyond the Psychometric Approach
edited by Wolfgang Schoppek, Joachim Funke, Magda Osman, Annette Kluge:

a look at the worlds' most pressing issues is helpful. Searching for strategies to cope with these problems is a difficult task: surely there is no place for the simple principle of "vary-one-thing-at-a-time" (VOTAT) when it comes to global problems. The VOTAT strategy is helpful in the context of simple problems (Wüstenberg et al., 2014); therefore, whether or not VOTAT is helpful in a given problem situation helps us distinguish simple from complex problems.

Because there exist no clear-cut strategies for complex problems, typical failures occur when dealing with uncertainty

The intermediate:


```
for m in range(1,6,1):
    for n in range(1,6,1):
        print(n, end='')
#-------------------------------
for m in range(1,6,1):
    for n in range(1,6,1):
        print(n, end='')
    print()
```

- Inner and outer loops
- Outer repeats 5 times
- Inner repeats 25 times

Output:

12345 12345 12345 12345 12345

12345

12345

12345

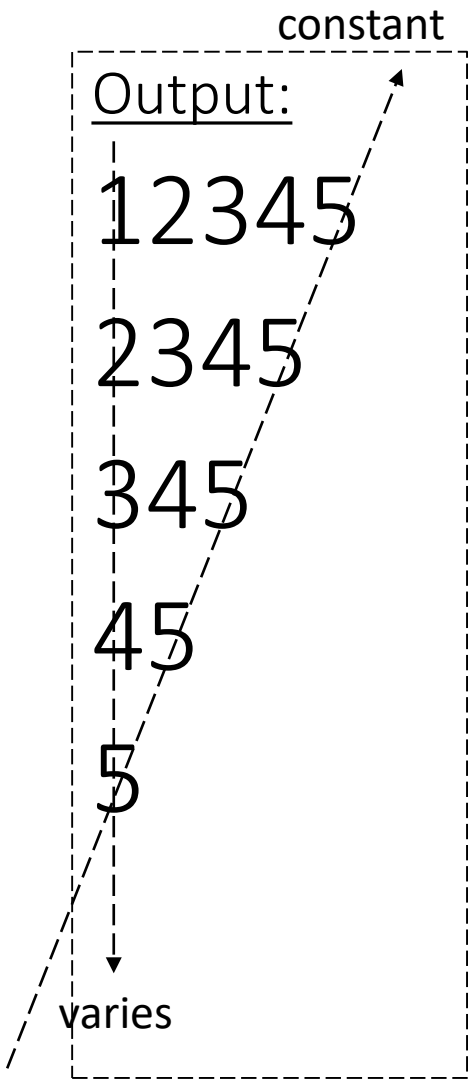
12345

12345

- What variable is printed?
- Row-by-row printing
- How many parameters or variables are involved?
- Identify the role of each parameter
- Parameter range can change
- What *for* works harder?
- Role of end='' (suppress newline)
- Role of print() (newline)

The hardest:

```
k=1
for m in range(1,6,1):
    for n in range(k,6,1):
        print(n,end='')
    k=k+1
    print()
```




```
#Simplified:
for m in range(1,6,1):
    for n in range(m,6,1):
        print(n,end='')
    print()
```

This may be your goal as solution to this problem

Sometimes:
print(n,end='',flush=True)

Nested Loops

Output:

1

22

333

4444

55555

Output:

11111

22222

33333

44444

55555

Output:

11111

or

55555

Nested Loops


```
i = 1
for j in range(1,6,1):
    print(i, end='')

# ----------------------------------------
for i in range(1,6,1):
    for j in range(1,6,1):
        print(i, end='')
    print()
```

Output:

11111

OUTPUT:

11111

22222

33333

44444

55555

FILE:nestedLoops01.py

Nested Loops: the hard one


```
for i in range(1,6,1):
    for j in range(1,i+1,1):
        print(i,end='')
    print()
```

Output:

```
1
22
333
4444
55555
```

FILE:nestedLoops01.py

Exercise: nested loops

A new application:

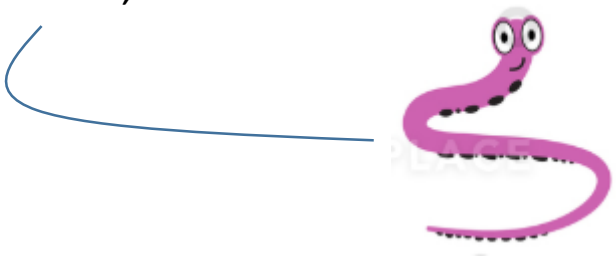
```
s=0
for i in range(1,6,1):
    for j in range(1,6,1):
        s=s+i
        print(i,"+ ",end='')
    print()
print("sum=",s)
```

```
1 + 1 + 1 + 1 + 1 +
2 + 2 + 2 + 2 + 2 +
3 + 3 + 3 + 3 + 3 +
4 + 4 + 4 + 4 + 4 +
5 + 5 + 5 + 5 + 5 +
sum= 75
```

FILE:nestedLoops01.py

How to learn Python

So If I attend 99%, I don't learn?



- Attend 100% this course
- Watch videos: youtube, etc.
- Read books (many of them free):
- Solve programming problems, many sites with plenty of exercises.
- Enroll into online classes, many sites, e.g.:
  - udemy.com
  - Cursera.org
  - Khan Academy (free)

QUIZ


```
for m in range(1,6,1):
    for n in range(1,6,1):
        if n==m:
            break
        print(n, end='')
    print()
```

What's the output: