15-110 Principles of Computing – F21

Lecture 10:
Debugging, Loops

Teacher:
Gianni A. Di Caro

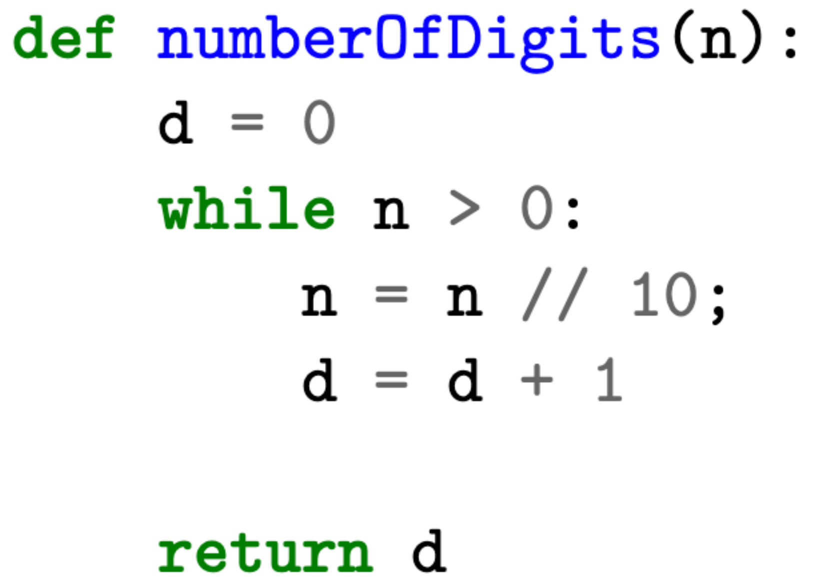
What the code does?

```
def [redacted](n):
    d = 0
    while n > 0:
        n = n // 10;
        d = d + 1

    return d
```

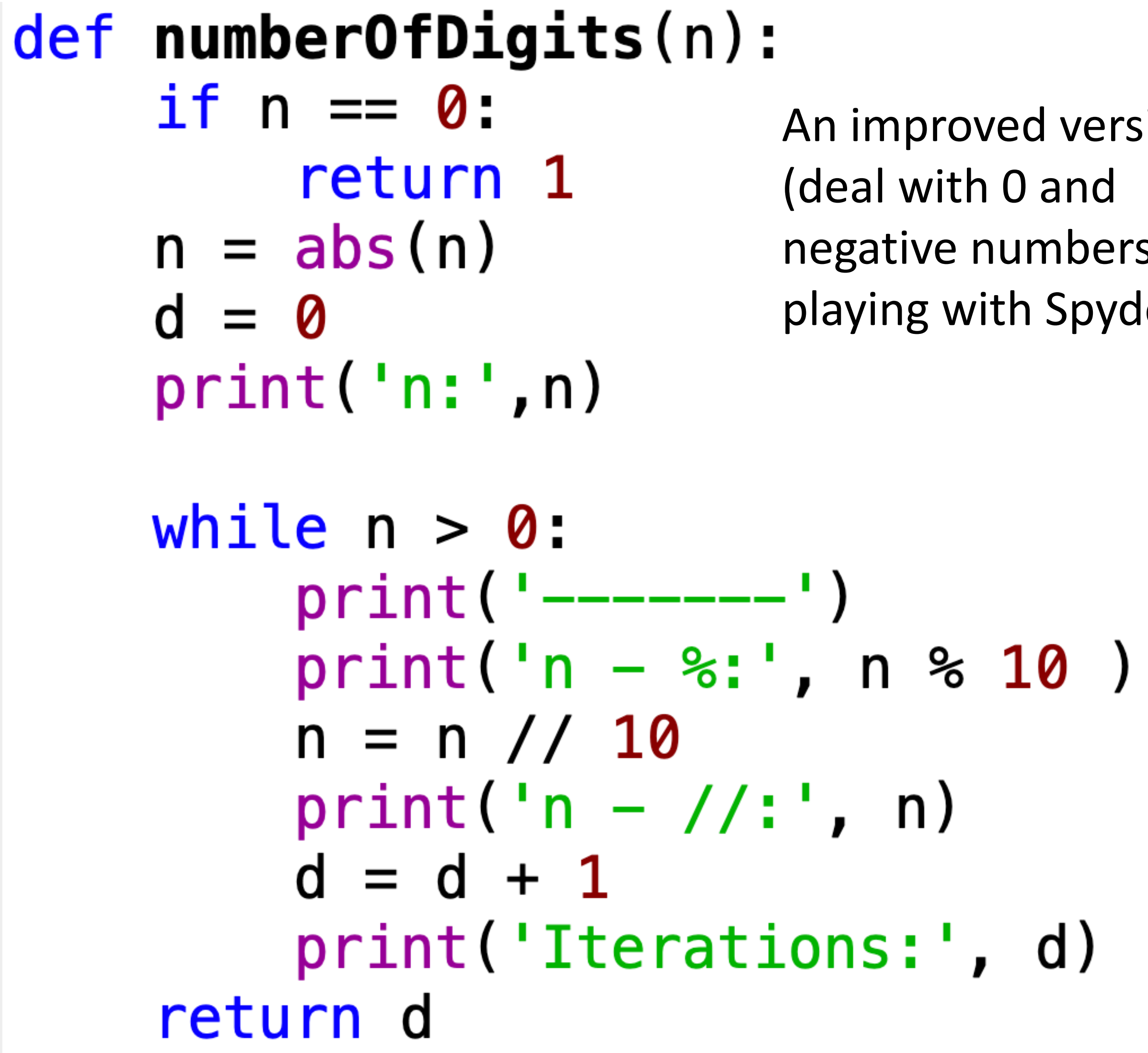
Let’s use `print()` to gain insights in the code!

What the code does? Use `print()` to get a better understanding!



```python
def numberOfDigits(n):
    d = 0
    while n > 0:
        n = n // 10;
        d = d + 1

    return d
```



```python
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    d = 0
    print('n:',n)

    while n > 0:
        print('--------')
        print('n - %:', n % 10 )
        n = n // 10
        print('n - //:', n)
        d = d + 1
        print('Iterations:', d)
    return d
```

An improved version (deal with 0 and negative numbers) after playing with Spyder

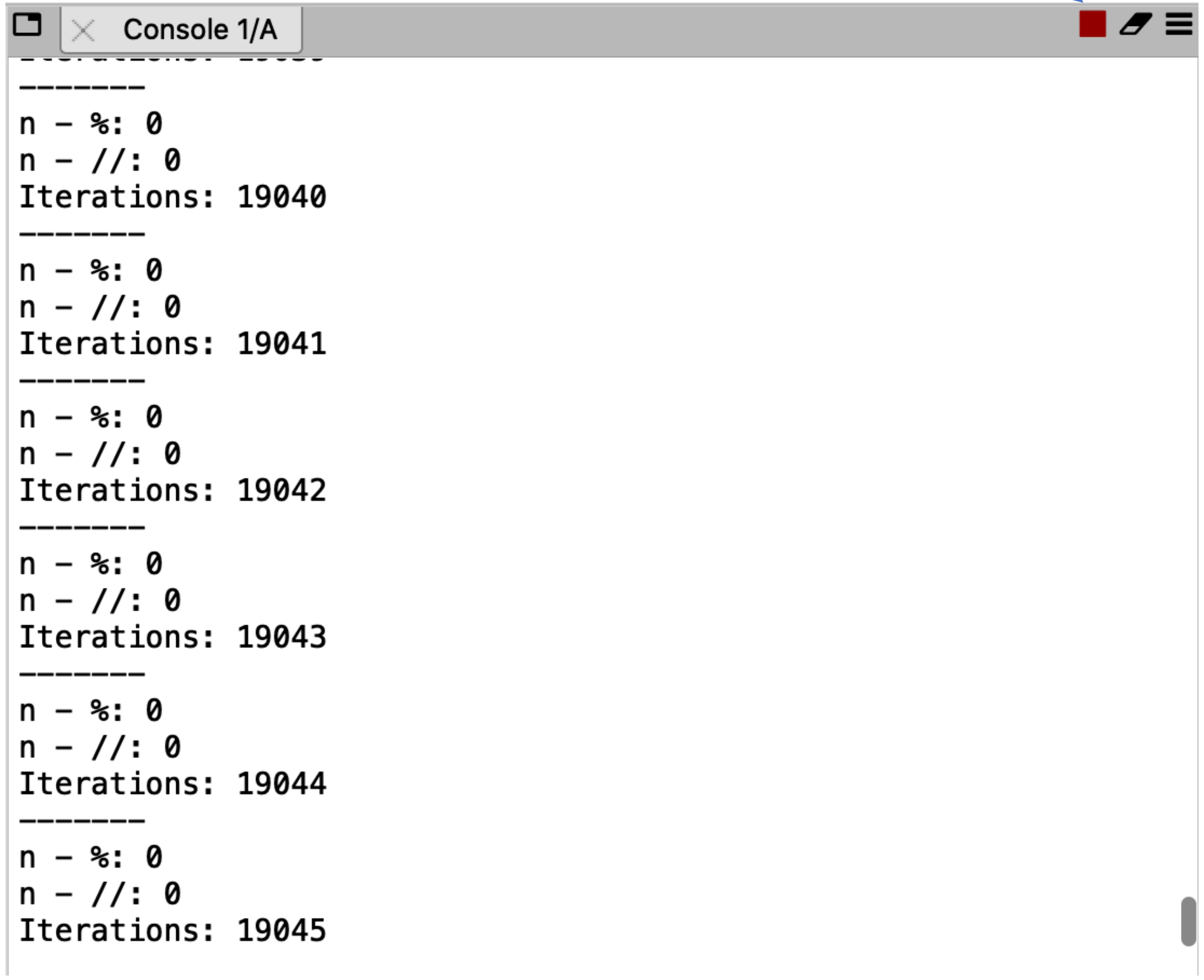
Forever looping → Interrupt the code with Spyder

This would loop forever!

Click here to interrupt a running code!

```python
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    d = 0
    print('n:',n)

    while n >= 0:
        print('-------')
        print('n - %:', n % 10 )
        n = n // 10
        print('n - //:', n)
        d = d + 1
        print('Iterations:', d)
    return d
```

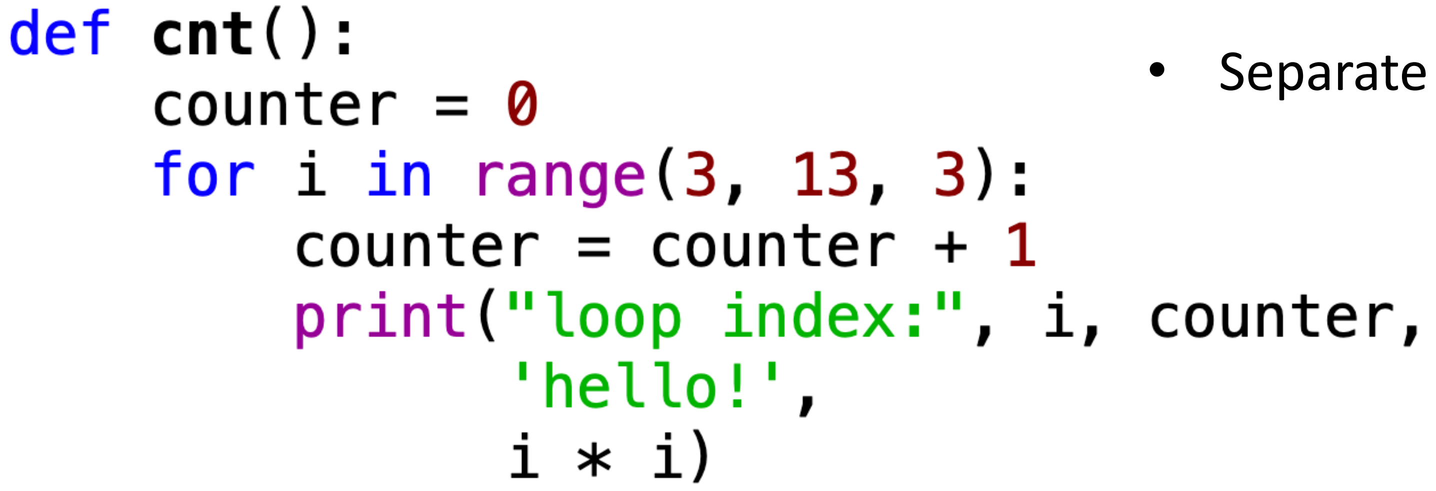


```
-------
n - %: 0
n - //: 0
Iterations: 19040
-------
n - %: 0
n - //: 0
Iterations: 19041
-------
n - %: 0
n - //: 0
Iterations: 19042
-------
n - %: 0
n - //: 0
Iterations: 19043
-------
n - %: 0
n - //: 0
Iterations: 19044
-------
n - %: 0
n - //: 0
Iterations: 19045
```

print( ) for online code debugging

```
counter = 0
for i in range(3, 13, 3):
   print('Loop index:', i)
   counter = counter + 1
print('Number of iterations:', counter, 'Final Loop index:', i )
```

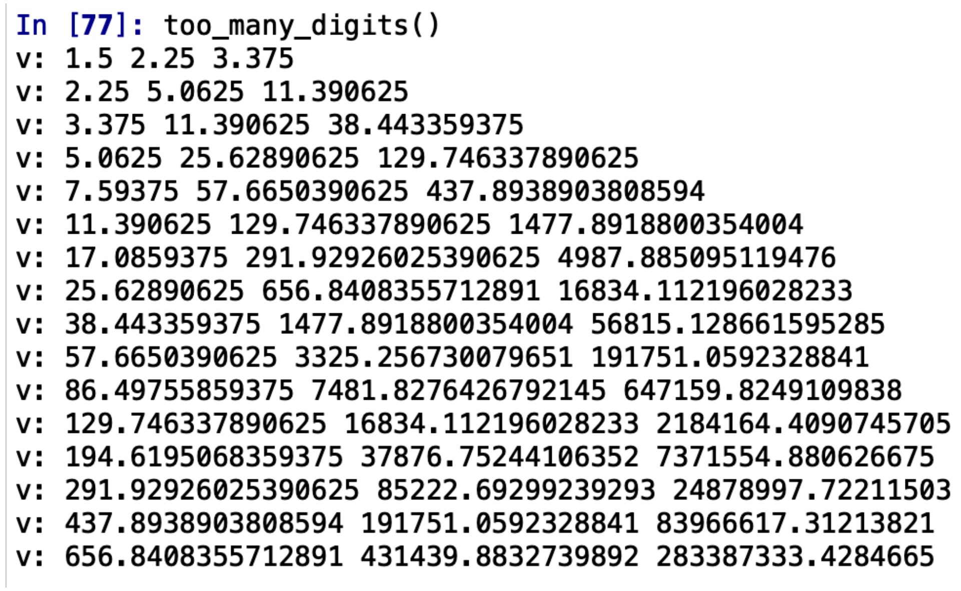
- Use print( ) to print anything useful to trace program behavior
- Separate items (string, variables) by commas



```
def cnt():
    counter = 0
    for i in range(3, 13, 3):
        counter = counter + 1
        print("loop index:", i, counter,
              'hello!',
              i * i)
```

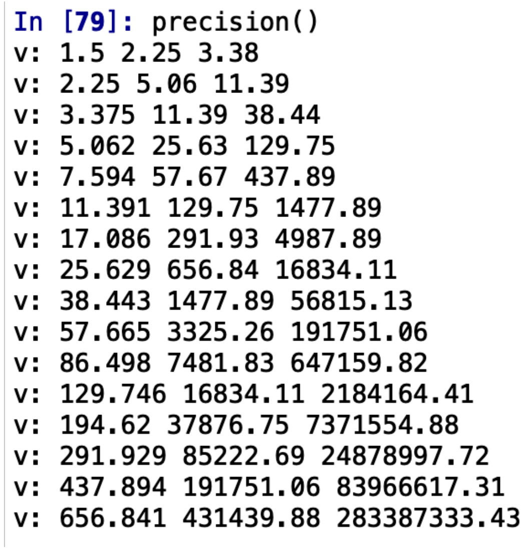
print( ) with round( )

```python
def too_many_digits():
    v = 1
    for i in range(3, 50, 3):
        v = v + ( v / 2)
        print("v:", v, v*v, v ** 3)
```



```
In [77]: too_many_digits()
v: 1.5 2.25 3.375
v: 2.25 5.0625 11.390625
v: 3.375 11.390625 38.443359375
v: 5.0625 25.62890625 129.746337890625
v: 7.59375 57.6650390625 437.8938903808594
v: 11.390625 129.746337890625 1477.8918800354004
v: 17.0859375 291.92926025390625 4987.885095119476
v: 25.62890625 656.8408355712891 16834.112196028233
v: 38.443359375 1477.8918800354004 56815.128661595285
v: 57.6650390625 3325.256730079651 191751.0592328841
v: 86.49755859375 7481.8276426792145 647159.8249109838
v: 129.746337890625 16834.112196028233 2184164.4090745705
v: 194.6195068359375 37876.75244106352 7371554.880626675
v: 291.92926025390625 85222.69299239293 24878997.72211503
v: 437.8938903808594 191751.0592328841 83966617.31213821
v: 656.8408355712891 431439.8832739892 283387333.4284665
```

```python
def precision():
    v = 1
    for i in range(3, 50, 3):
        v = v + ( v / 2)
        print("v:", round(v,3),
              round(v*v, 2), round(v**3, 2))
```



```
In [79]: precision()
v: 1.5 2.25 3.38
v: 2.25 5.06 11.39
v: 3.375 11.39 38.44
v: 5.062 25.63 129.75
v: 7.594 57.67 437.89
v: 11.391 129.75 1477.89
v: 17.086 291.93 4987.89
v: 25.629 656.84 16834.11
v: 38.443 1477.89 56815.13
v: 57.665 3325.26 191751.06
v: 86.498 7481.83 647159.82
v: 129.746 16834.11 2184164.41
v: 194.62 37876.75 7371554.88
v: 291.929 85222.69 24878997.72
v: 437.894 191751.06 83966617.31
v: 656.841 431439.88 283387333.43
```

What the code does?

```
def [redacted](n):
    p = 0
    while (n % (10 ** p)) != n:
        p = p + 1

    return p
```

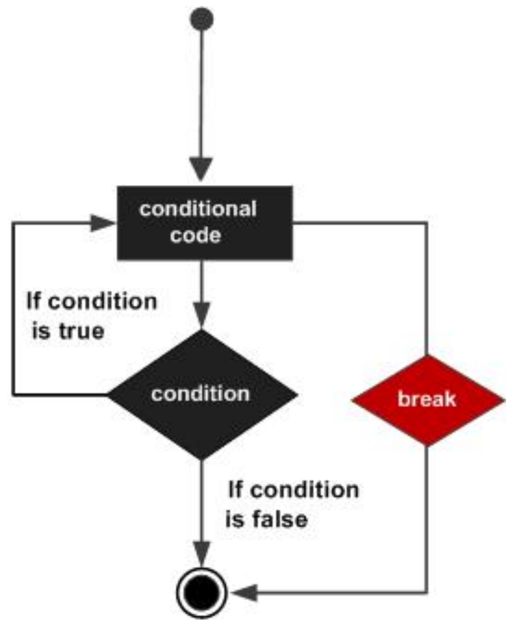
Two equivalent ways: while condition vs. while True + break

```
def numberOfDigits(n):
    p = 0
    while (n % (10 ** p)) != n:
        p = p + 1

    return p
```

The same, but more decomposed

```
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    p = 0
    while True:
        powers10 = 10 ** p
        print('powers of 10:', powers10, 'p:',p)
        if ( n % powers10 ) == n:
            print('Break at ', p)
            break
        else:
            p = p +1
    return p
```



break vs. return

```python
def numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    p = 0
    while True:
        powers10 = 10 ** p
        print('powers of 10:', powers10, 'p:',p)
        if ( n % powers10 ) == n:
            print('Break at ', p)
            break
        else:
            p = p +1
    return p
```

```python
ef numberOfDigits(n):
    if n == 0:
        return 1
    n = abs(n)
    p = 0
    while True:
        powers10 = 10 ** p
        print('powers of 10:', powers10, 'p:',p)
        if ( n % powers10 ) == n:
            print('Break at ', p)
            return p
        else:
            p = p +1
```

Do the same thing: break and return, or directly return

From Lab03

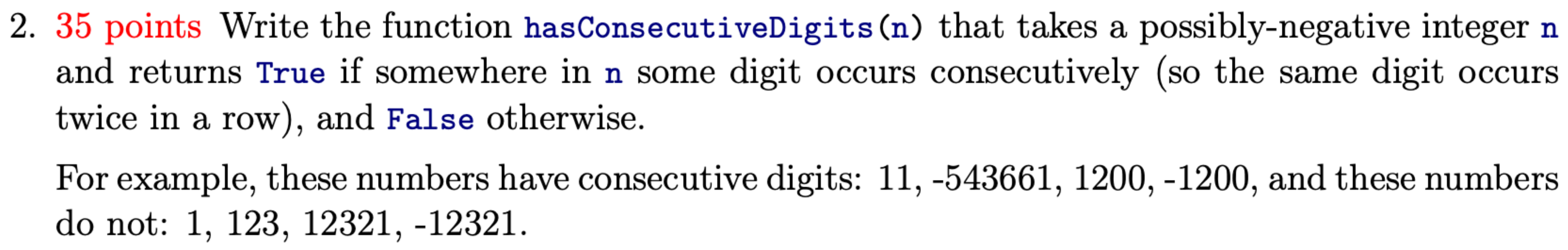
30 points **Prime numbers**

A *prime number* is a number that is divisible only by two distinct numbers: 1 (one) and by itself. For example the number 7 is Prime, because it can be divided only by 1 and by 7.

Implement the function `isPrime(n)` that returns `True` if `n` is a prime number, or `False` otherwise.

```
def isPrime(n):
    if n == 0 or n == 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

From Lab04



2. 35 points Write the function `hasConsecutiveDigits(n)` that takes a possibly-negative integer `n` and returns `True` if somewhere in `n` some digit occurs consecutively (so the same digit occurs twice in a row), and `False` otherwise.

For example, these numbers have consecutive digits: 11, -543661, 1200, -1200, and these numbers do not: 1, 123, 12321, -12321.

```
def hasConsecutiveDigits(n):
    if n < 0:
        n = -n

    last = -1
    while n > 0:
        if n%10 == last:
            return True
        last = n%10
        n //= 10
    return False
```

Use print() to understand what's going on!