Lecture 2

- 4.1 For Loops
- 4.2 Case Study: DNA to RNA Transcription
- 5.1 While loops

4.1 For Loops

- use when we need to repeat a series of steps [for each item in a sequence]

Syntax of the *for loop*:

```
for identifier in sequence:
    body
```

Example 2:

```
for i in range(10):

    t = i*10
    print(“iteration”,i,”:”,t)
```

see program for-loop.py

4.1 For Loops

One more example (see *for-list.py* for full version):

```
groceries = ["Milk","Sugar","Bread","Honey"]

enum=1
for item in groceries:
    item=str(enum) + '. ' + item
    enum +=1
    print(item)
```

Can you predict what the program will do?

4.1 For Loops

Types of loops:

- definite loops

  example: for i in range(5)

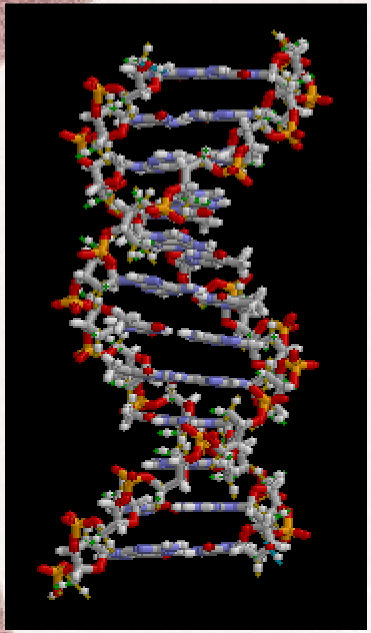
- index-based loops (counted loops)

  example: for i in range(len(guestList)

- nested loops

  example:
```
for i in range(5):
    for j in range(1):
        body of the loop
```

DNA and RNA



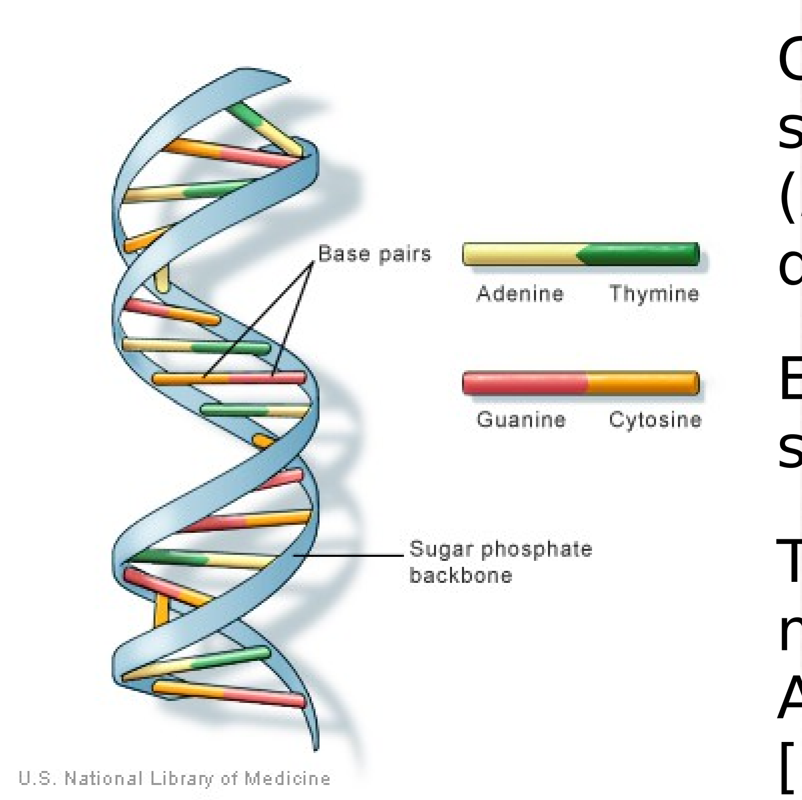
DNA stands for Deoxyribonucleic acid. It is a *nucleic acid* that contains the genetic instructions used in the development and functioning of all known living organisms and some viruses.

The main role of DNA molecules is the *long-term storage of information*.

The DNA segments that carry this genetic information are called genes.

Other DNA sequences have structural purposes, or are involved in regulating the use of this genetic information.

DNA and RNA



Chemically, DNA consists of two long strands (*polymers*) of simple units (*nucleotides*) that run in opposite directions to each other.

Each strand has *backbone* made of sugars and phosphate groups.

To each sugar one of four types of molecules called bases is attached: Adenine (A) ['ædənɪn], Cytosine (C) ['saɪtəsɪn], Guanine (G) ['guːəˌniːn], or Thymine (T) ['thī-ˌmēn]

The sequence of these four bases along the backbone encodes the information.

Each type of base on one strand forms a bond with just one type of base on the other strand: A bonds only to T, and C bonds only to G.

DNA and RNA

RNA stands for Ribonucleic acid. It is also a *nucleic acid,* as DNA, and is used to create proteins.

And is very similar to DNA, but RNA is usually single-stranded, while DNA is usually double-stranded. There are other differences between DNA and RNA, but we won't mention them here.

Organisms use DNA as a model when constructing a RNA.

The process of creating RNA from DNA is called *transcription*.

RNA consists of four nucleotides:
Adenine (A) [ˈædənɪn],
Cytosine (C) [ˈsaɪtəsɪn],
Guanine (G) [ˈguːəˌniːn], and
Uracil (U) [y r'ə-sĭl]

CSI 32

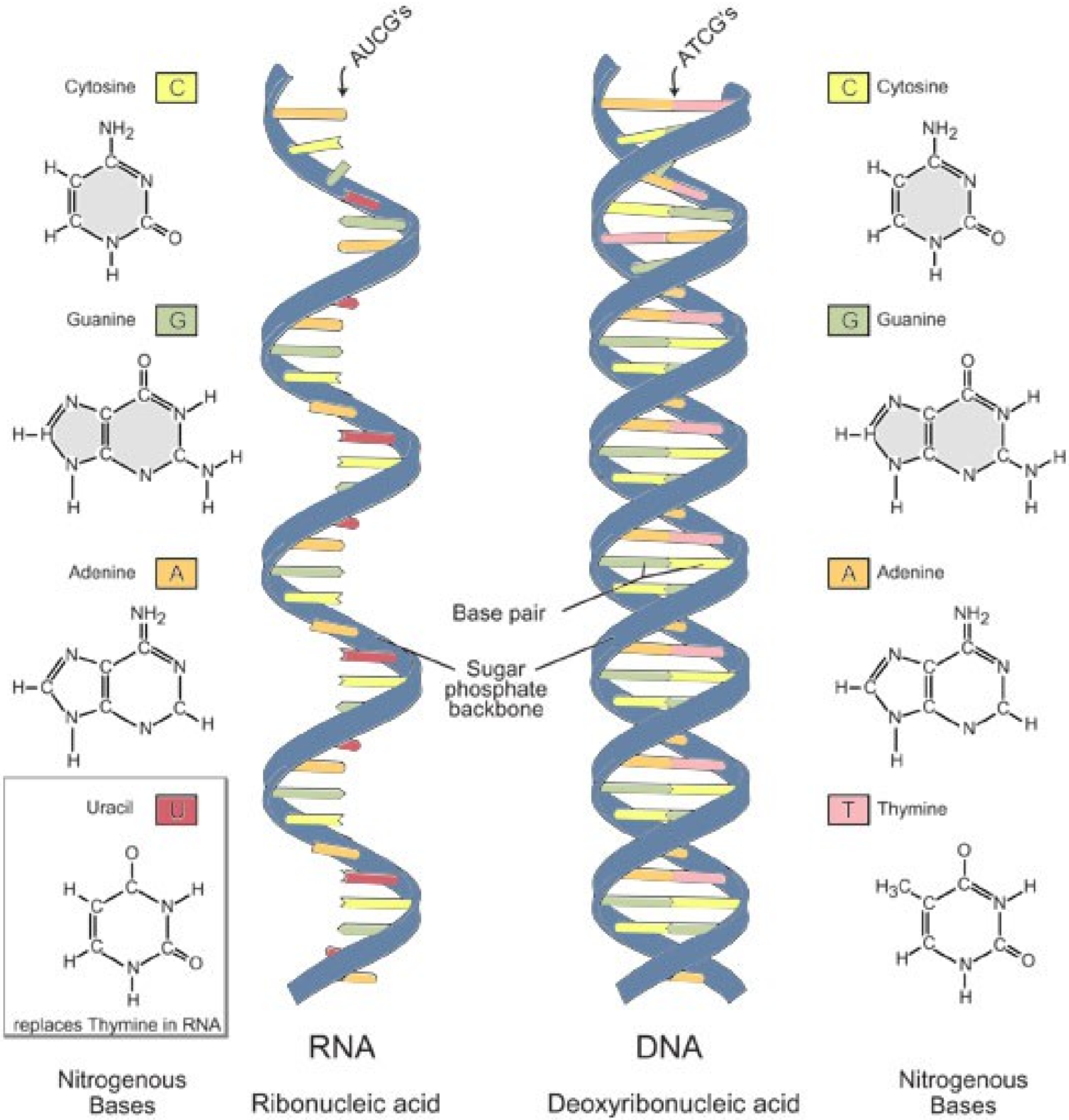


Image adapted from: National Human Genome Research Institute. Talking Glossary of Genetic Terms. Available at: www.genome.gov/ Pages/Hyperion//DIR/VIP/Glossary/Illustration/rna.shtml.

4.2 Case Study: DNA to RNA Transcription

Transcription creates an RNA sequence by matching a complementary base to each original base in DNA, using following substitutions:

DNA		RNA
A (Adenine)	→	U (Uracil)
C (Cytosine)	→	G (Guanine)
G (Guanine)	→	C (Cytosine)
T (Thymine)	→	A (Adenine)

Let's write a program that will do the transcription.

DNA to RNA Transcription

Analysis:

Let's discuss the requirements:

input

1. A user will be prompted to input a DNA sequence

2. He/she should get RNA sequence that will be built from the DNA.

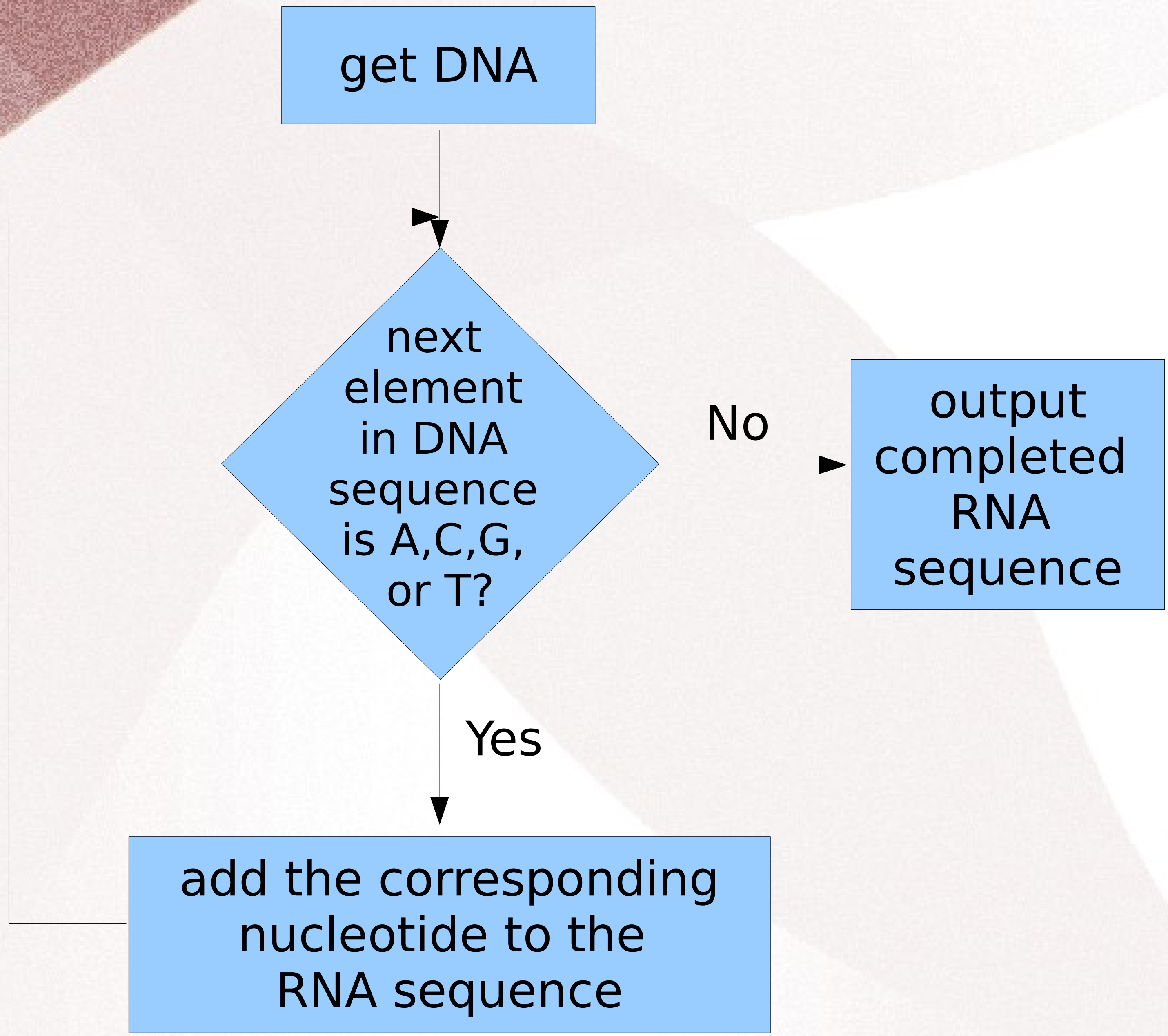
output + what the program should do

Design of the program:

We are not building/defining any new classes; we will use string class.

Let's present Activity Diagram (flowchart) for our program.

DNA to RNA Transcription



DNA to RNA Transcription

The sketch of our program:

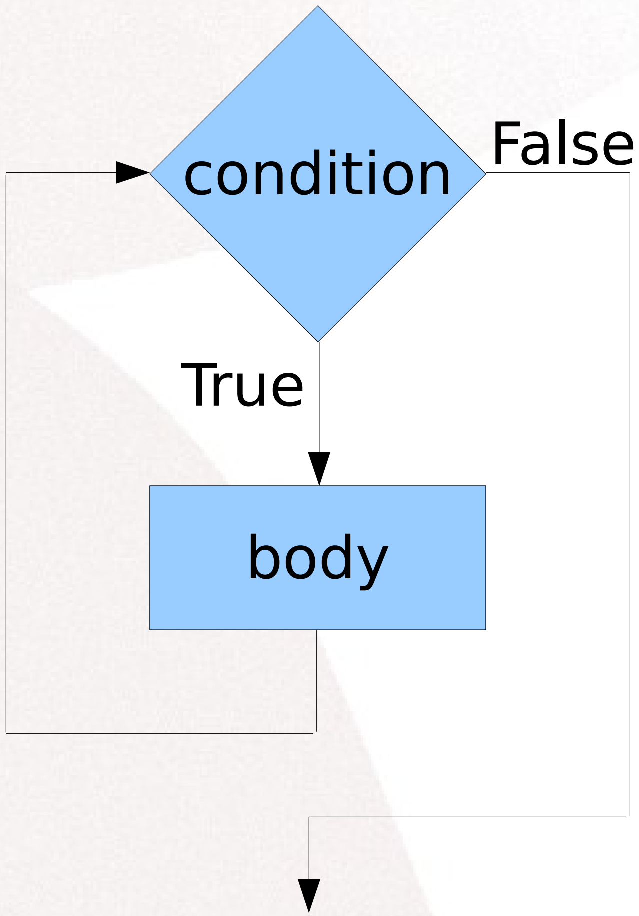
```
def hello():
    ...

def transcribe(dna):
    ...

def main()
    get input, assign to dna

    result=transcribe(dna)

    output result

main()
```

See the implementations in
DNA-RNA-trans.py, DNA-RNA-trans2.py

5.1 While loops

Syntax:

```
while condition:
    body
```



In any loop body the command **break** causes an immediate stop to the entire loop.
(*very useful sometimes*)

Example:

```
while response.lower() in ('y','yes')
    # body of the loop:
    …
    response=input("Continue (Yes/No)?

# out of the loop code:
…
```

5.1 While loops

Infinite loops

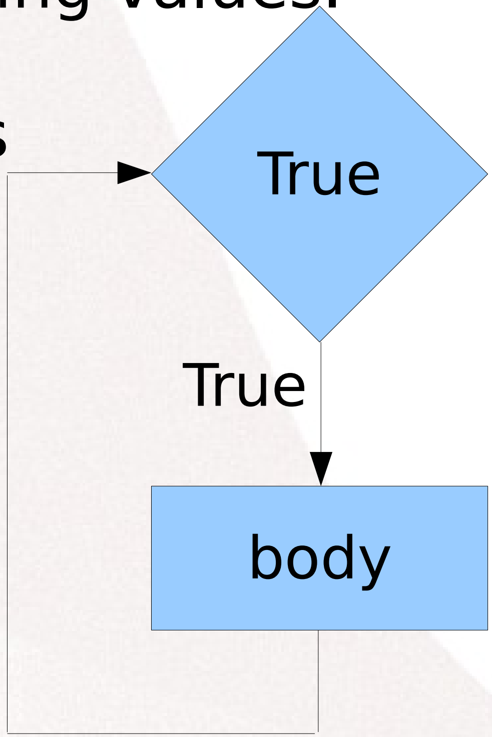
In a for loop, the overall number of iterations is naturally bounded based on the length of the original sequence.

In a while loop, the overall number of iterations is not explicitly bounded. It is determined by a combination of the loop condition and the changing state of underlying values.

A potential pitfall: the while loop never ends (infinite loop)

Example:

```
while True:
   print("Hello!")
```



5.1 While loops

Example:

An integer **k** ≥ **2** is a **prime number** if it is not evenly divisible by any numbers in **range(2,k)**. Write a program that checks whether a given number **n** is a prime number or a composite number.

5.1 While loops

Example:
An integer **k** ≥ **2** is a **prime number** if it is not evenly divisible by any numbers in **range(2,k)**. Write a program that checks whether a given number **n** is a prime number or a composite number.

A comment:
When checking whether a number **n** is prime or not, there is no need to check the divisibility of this number by a number, whose square is more than **n**.
- this reduces the complexity of an algorithm

5.1 While loops

Example:
An integer **k** ≥ **2** is a **prime number** if it is not evenly divisible by any numbers in **range(2,k)**. Write a program that checks whether a given number **n** is a prime number or a composite number.

A comment:
When checking whether a number **n** is prime or not, there is no need to check the divisibility of this number by a number, whose square is more than **n**.
- this reduces the complexity of an algorithm.

Question: what numbers we don't need to check for primality?

5.1 While loops

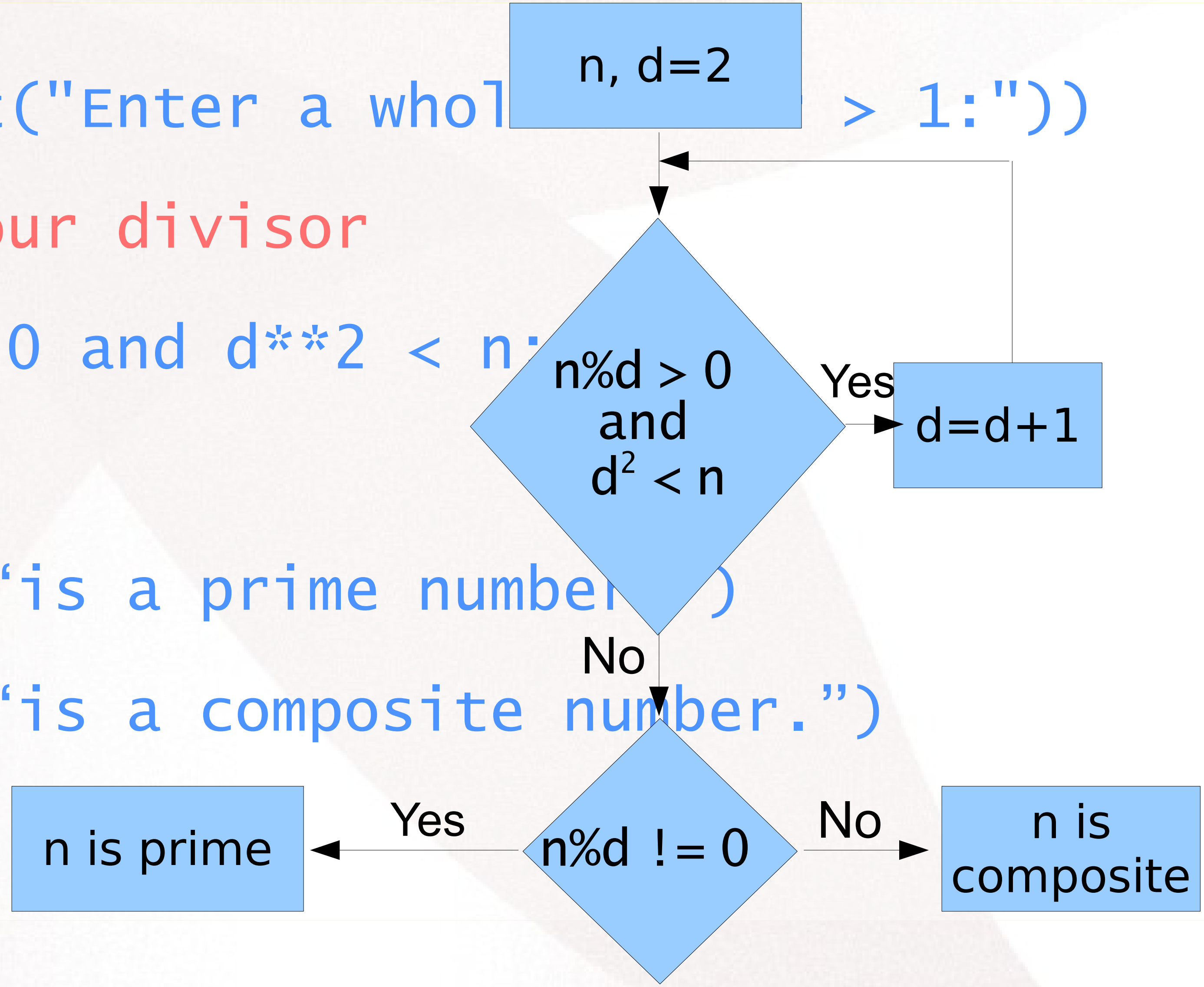
```
n=eval(input("Enter a whole number > 1:"))

d=2 # d is our divisor

while n%d > 0 and d**2 < n:
   d+=1

if n%d != 0:
   print(n, “is a prime number.”)
else:
   print(n, “is a composite number.”)
```

Two exit conditions.
What are they?

see the program while-loop.py

5.1 While loops

```
n=eval(input("Enter a whol            > 1:"))

d=2 # d is our divisor

while n%d > 0 and d**2 < n:
    d+=1

if n%d != 0:
    print(n, “is a prime numbe     )
else:
    print(n, “is a composite number.”)
```



In-class Work

- Write a program that generates multiplication table *nxn* for a positive integer *n*
- Consider the following code:

```
a = 10
b = 64
while a < b:
        print(a,” “,b)
        a -= 1
        b = b/2
```

Predict the output of this code if executed

Homework 2 assignment

Exercise 1: give a flowchart portraying the algorithm you wrote for problem 1.5 or 1.8

Page 155 / exercise 4.29

Page 196 / exercise 5.6