

1. The Assignment Statement and Types

Topics:

Python's Interactive Mode

Variables

Expressions

Assignment

Strings, Ints, and Floats

The Python Interactive Shell

Python can be used in a way that reminds you of a calculator. In the `` command shell of your system simply type

```
python
```

and you will be met with a prompt...

```
>>>
```

Let's Compute the Area of a Circle Using Python

```
>>> r = 10
>>> A = 3.14*r*r
>>> print (A)
314.0
```

Programming vs Math

```
>>> r = 10
>>> A = 3.14*r*r
>>> print (A)
314.0
```

Notation is different.

In Python, you can't say $A = 3.14xr^2$

Programming vs Math

```
>>> r = 10
>>> A = 3.14*r**2
>>> print (A)
314.0
```

Notation is different.

In Python you indicate exponentiation with **

Programming vs Math

```
>>> r = 10
>>> A = 3.14*r**2
>>> print (A)
314.0
```

r and A are **variables**. In algebra, we have the notion of a variable too. But there are some big differences.

Variables

```
>>> r = 10  
>>> A = 3.14*r**2
```

r -> 10

A -> 314.0

A variable is a named memory location. Think of a variable as a box.

It contains a value. Think of the value as the contents of the box.

" The value of r is 10. The value of A is 314.0."

The Assignment Statement

```
>>> r = 10
```

r -> 10

The "=" symbol indicates assignment.

The assignment statement `r = 10` creates the variable `r` and assigns to it the value of 10.

Formal: "r is assigned the value of 10"

Informal: "r gets 10"

The Assignment Statement

```
>>> r = 10
>>> A = 3.14*r**2
```

r -> 10 A -> 314.0

A variable can be used in an **expression** like
`3.14*r**2`.

The expression is evaluated and then stored.

Assignment Statement: WHERE TO PUT IT = RECIPE FOR A VALUE

Order is Important

```
>>> A = 3.14*r**2
>>> r = 10
NameError: name 'r' is not defined
```

Math is less fussy:

$$A = 3.14*r**2 \text{ where } r = 10$$

Assignment vs. "Is Equal to"

```
>>> r = 10
```

```
>>> 3.14*r**2 = A
```

```
SyntaxError: can't assign to an  
operator
```

In Math "=" is used to say what is on the left equals what is on the right.

In Python, "=" prescribes an action, "evaluate the expression on the right and assign its value to the variable named on the left."

The Assignment Statement

```
>>> r = 10
>>> A = 3.14*r**2
>>> S = A/2
```

```
r -> 10
A -> 314.0
S -> 157.0
```

Here we are assigning to `S` the area of a semicircle that has radius 10.

The Key 2-Step Action Behind Every Assignment Statement

< variable name > = < expression >

1. Evaluate the expression on the right hand side.
2. Store the result in the variable named on the left hand side.

Naming Variables

```
>>> radius = 10
>>> Area = 3.14*radius**2
```

radius -> 10

Area -> 314.0

Rule 1. Name must be comprised of digits, upper case letters, lower case letters, and the underscore character “_”

Rule 2. Must begin with a letter or underscore

A good name for a variable is short but suggestive of its role: `Circle_Area`

Precedence

- Q. In an arithmetic expression, what is the order of evaluation?
- A. Exponentiation & negation comes before multiplication & division which in turn come before addition & subtraction.

This:

$$A + B * C$$

$$-A ** 2 / 4$$

$$A * B / C * D$$

Is the same as:

$$A + (B * C)$$

$$- (A ** 2) / 4$$

$$((A * B) / C) * D$$

It is a good habit to use parentheses if there is the slightest ambiguity.

Revisit Circle Area

```
>>> r = 10
>>> A = (22/7)*r**2
>>> print (A)
300.0
```

It seems that Python evaluates $(22/7)$ as 3 instead of 3.142... WHY?

A different kind of arithmetic. We have a related experience here.
 $11+3 = 2$ in "clock arithmetic"

Integers and Decimals

In math we distinguish between integer numbers and decimal numbers.

Integer Numbers:

100, 0, -89, 1234567

Decimal Numbers:

-2.1, 100.01, 100.0, 12.345

Integers and Decimals

There are different kinds of division.

Integer Division:

$30/8$ is 3 with a remainder of 6

Decimal Division:

$30/8$ is 3.75

int vs float

In Python, a number has a **type**.

The **int** type represents numbers as integers.

The **float** type represents numbers as decimals.

Int and Float Arithmetic

```
>>> x = 30
>>> y = 8
>>> q = x/y
>>> print (q)
3.75
>>> r = x%y
>>> print (r)
6
```

To get the remainder, use %. Python "knows" that the values stored in x and y have type int because there are no decimal points in those assignments.

Strings

So far we have discussed computation with numbers.

Now we discuss computation with text.

We use **strings** to represent text.

You are a "string processor" when you realize $7/4$ means July 4 and not 1.75!

Strings

Strings are quoted characters. Here are three examples:

```
>>> s1 = 'abc'  
>>> s2 = 'ABC'  
>>> s3 = ' A B C '
```

`s1`, `s2`, and `s3` are variables with string value.

Strings

Nothing special about letters...

```
>>> Digits = '1234567890'  
>>> Punctuation = '! : ; . ?'  
>>> Special = '@#$%^&* () _ - + = \'
```

Basically any keystroke but there are some exceptions and special rules. More later.

Here is one: `'Sophie'' 's Choice'` i.e., Sophie's Choice

Strings are Indexed

```
>>> s = 'The Beatles'
```

s -->

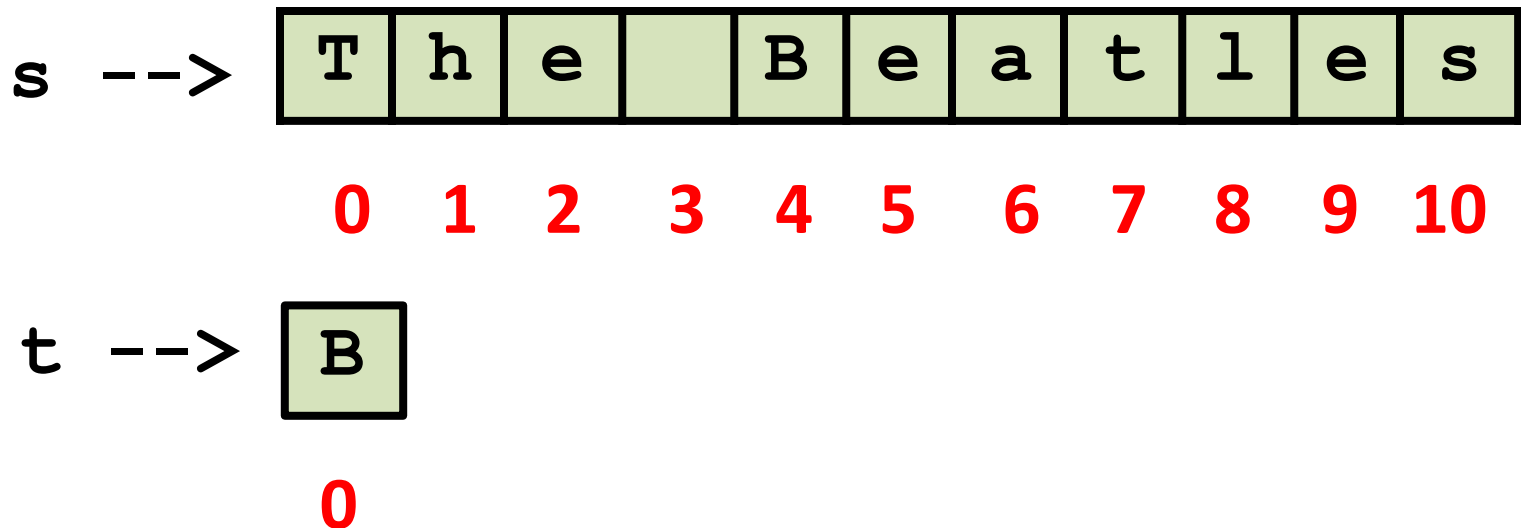
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|
| T | h | e | | B | e | a | t | l | e | s |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

The characters in a string can be referenced through their indices. Called "subscripting".

Subscripting from zero creates a disconnect: 'T' is not the first character.

Strings are Indexed

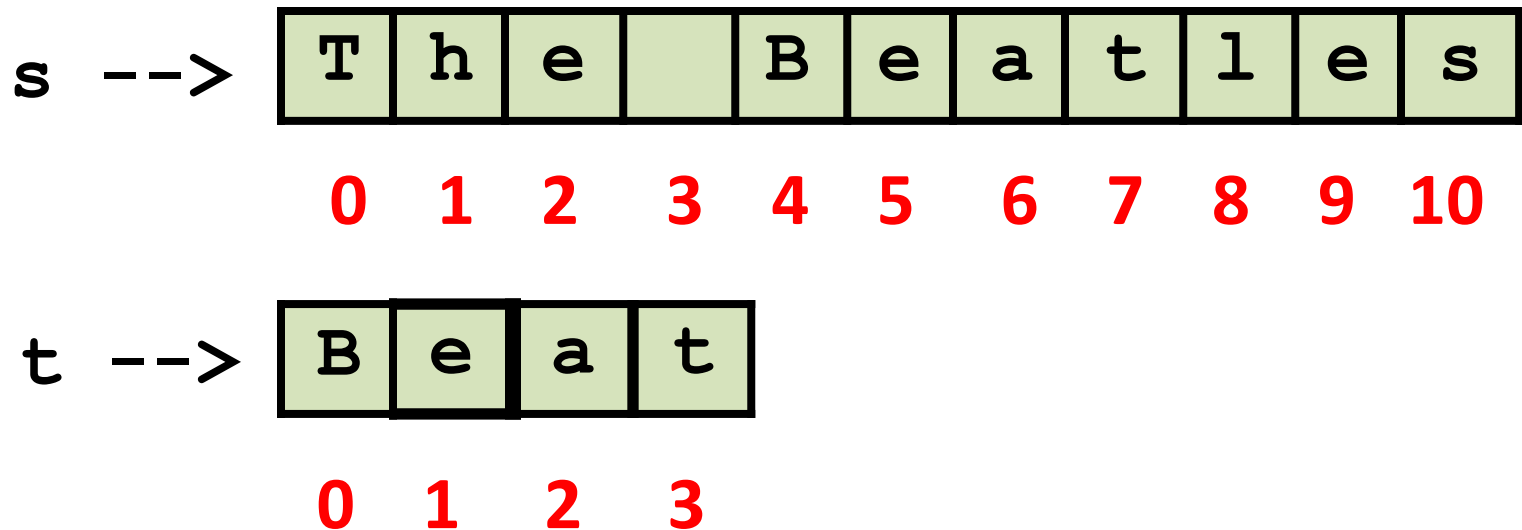
```
>>> s = 'The Beatles'  
>>> t = s[4]
```



The square bracket notation is used. Note, a single character is a string.

String Slicing

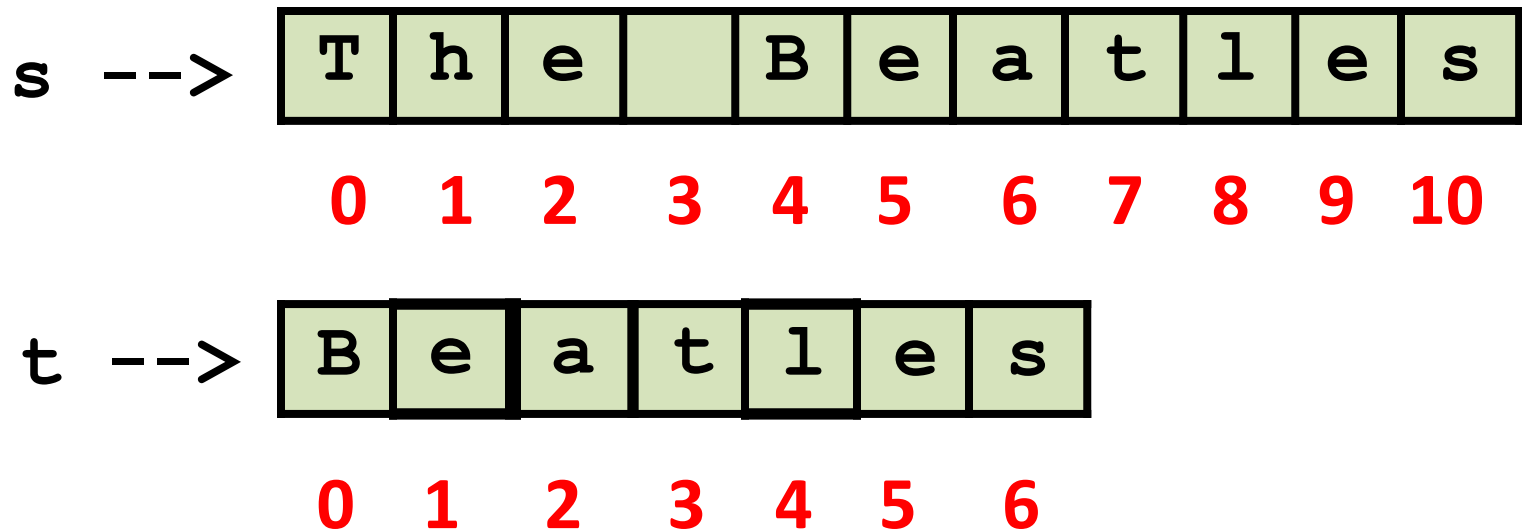
```
>>> s = 'The Beatles'  
>>> t = s[4:8]
```



We say that "t is a slice of s".

String Slicing

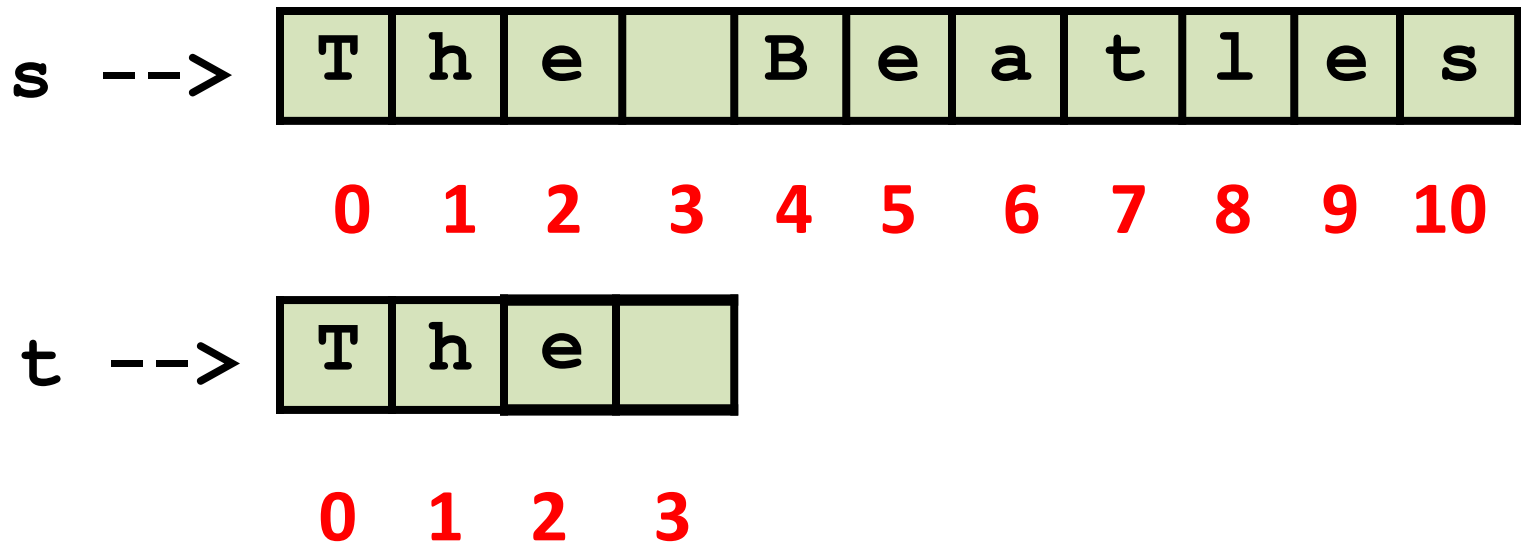
```
>>> s = 'The Beatles'  
>>> t = s[4:]
```



Same as `s[4:11]`. Handy notation when you want an "ending slice."

String Slicing

```
>>> s = 'The Beatles'  
>>> t = s[:4]
```



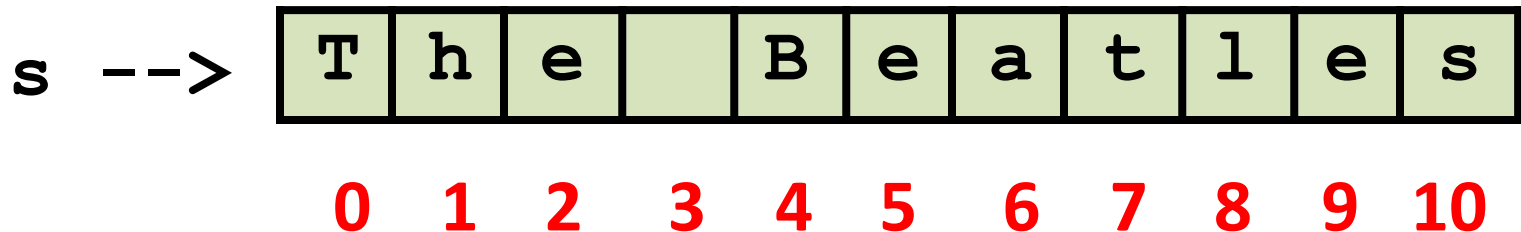
Same as `s[0:4]`. Handy notation when you want a "beginning slice".

String Slicing

```
>>> s = 'The Beatles'
```

```
>>> t = s[11]
```

```
IndexError: string index out of  
range
```

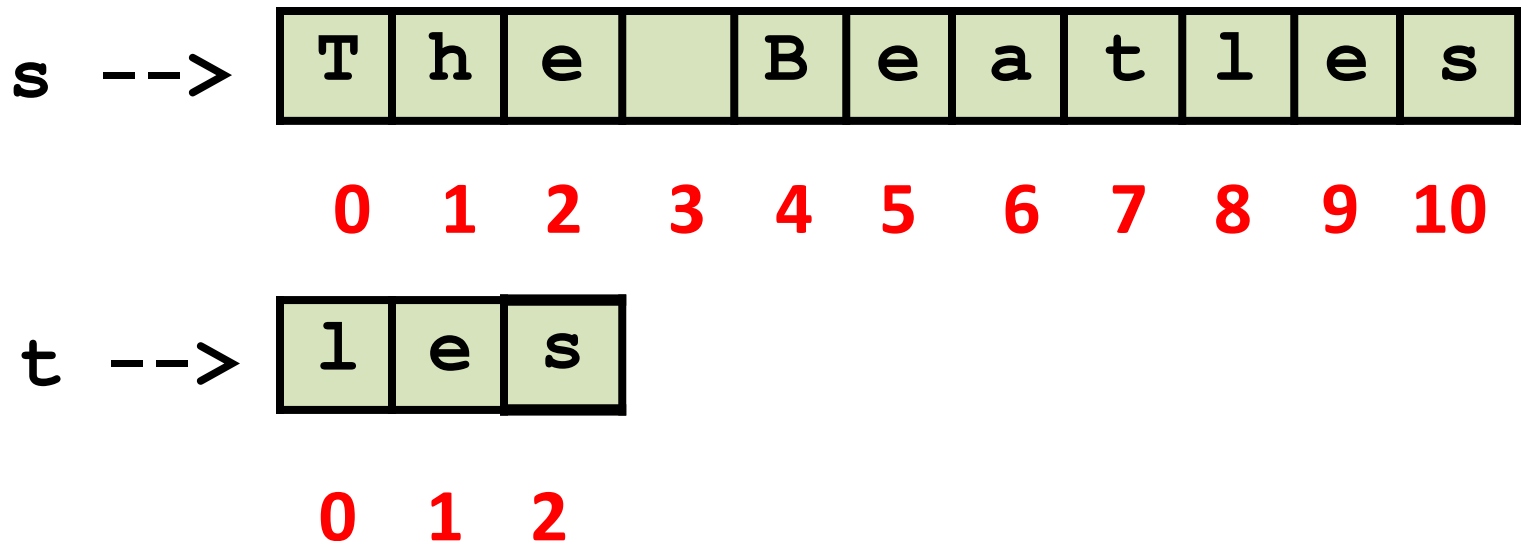


There is no s[11]. An illegal to access.

Subscripting errors are **EXTREMELY** common.

String Slicing

```
>>> s = 'The Beatles'  
>>> t = s[8:20]
```



It is "OK" to shoot beyond the end of the source string.

Strings Can Be Combined

```
>>> s1 = 'The'  
>>> s2 = 'Beatles'  
>>> s = s1+s2
```

s -->

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| T | h | e | B | e | a | t | l | e | s |
|---|---|---|---|---|---|---|---|---|---|

This is called **concatenation**.

Concatenation is the string analog of addition except

Concatenation

```
>>> s1 = 'The'  
>>> s2 = 'Beatles'  
>>> s = s1 + ' ' + s2
```

s -->

| | | | | | | | | | | |
|---|---|---|--|---|---|---|---|---|---|---|
| T | h | e | | B | e | a | t | l | e | s |
|---|---|---|--|---|---|---|---|---|---|---|

We "added" in a blank.

No limit to the number of input strings: $s = s_1 + s_2 + s_3 + s_4 + s_5$

Types

Strings are a type: `str`

So at this point we introduced 3 types:

`int` for integers, e.g., `-12`

`float` for decimals, e.g., `9.12`, `-12.0`

`str` for strings, e.g., `'abc'`, `'12.0'`

A Type is a Set of Values and Operations on Them

Operations...

int + - * / unary- ** %

float + - * / unary- **

str +



concatenation

Type Conversion

```
>>> s = '123.45'  
>>> x = 2*float(s)  
>>> print(x)  
246.90
```

A string that encodes a decimal value can be represented as a `float`.

Type Conversion

```
>>> s = '-123'  
>>> x = 2*int(s)  
>>> print (x)  
-246
```

A string that encodes an integer value can be represented as an `int`.

Type Conversion

```
>>> x = -123.45
>>> s = str(x)
>>> print (s)
'-123.45'
```

Shows how to get a string encoding of a float value.

Summary

1. Variables house values that can be accessed.
2. Assignment statements assign values to variables.
3. Numerical data can be represented using the `int` and `float` types.
4. Text data can be represented using the `str` type.