



Intro to Python

IOE 373 Lecture 13



Basic Elements

- Background
- Elements of Python
- Variables, Expressions and Statements
- Conditional Execution




Python

- Developed by Guido Van Rossum in 1989
- General purpose, open source and free
- Very popular with data scientists
- Many libraries (NumPy, SciPy, Matplotlib, Pandas, etc.)
- Dictionaries – enable fast database-like operations
 - Great for text analysis (e.g. social media)

Why is it named Python?

- While implementing Python, Van Rossum was also reading the published scripts from “Monty Python’s Flying Circus”, a BBC comedy series from the 1970s. Since he wanted a short, unique and slightly mysterious name for his invention, he got inspired by the series and named it Python!





Python's Core Philosophy (The Zen of Python)

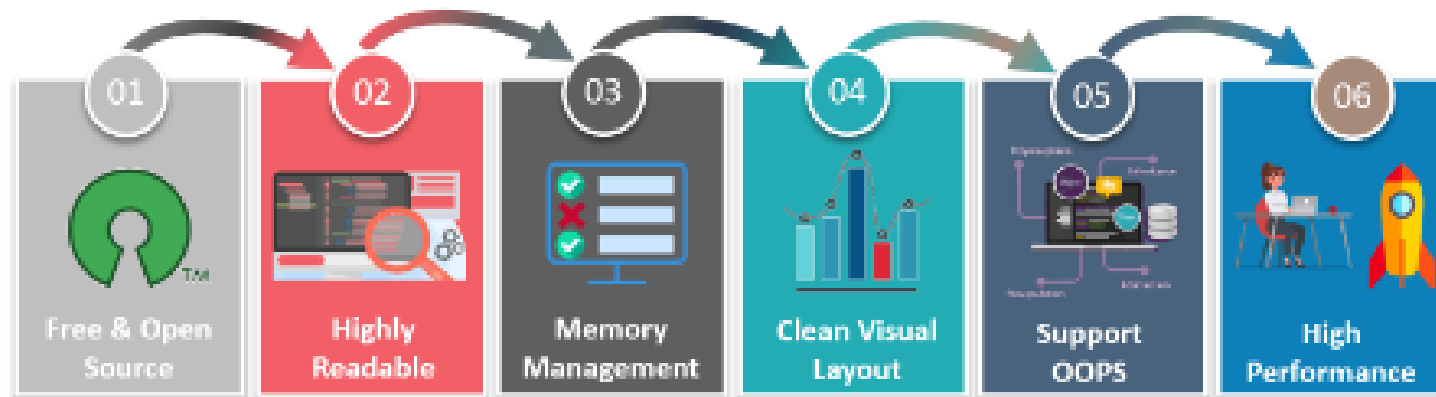
- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Python Domains



Source: Edureka.co

Python Users and Features



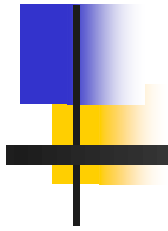
Source: Edureka.co



Elements of Python

- Vocabulary / Words - Variables and Reserved words
- Sentence structure - valid syntax patterns
- Story structure - constructing a program for a purpose
- I like using the Anaconda Suite (it's free!!):
<https://www.anaconda.com/distribution/>
- Download your own copy or use it on CAEN

Sample Python code (counting common words)



```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

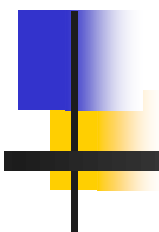
print("Most common word is: ",bigword)
print("Count is: ",bigcount)
```

A short “story”
about how to count
words in a file in
Python

python wordcount.py

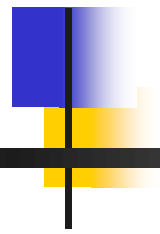
Enter file: words.txt

Most common word is: to
Count is: 16



Sentences or Lines

<code>x = 2</code>	←	Assignment statement
<code>x = x + 2</code>	←	Assignment with expression
<code>String="Hello"</code>		
<code>print(x)</code>	←	Print statement



Python Scripts

- Interactive Python is good for experiments and programs of 3-4 lines long.
- Most programs are much longer, so we type them into a file and tell Python to run the commands in the file.
- In a sense, we are “giving Python a script”.
- As a convention, we add “.py” as the suffix on the end of these files to indicate they contain Python.
 - Jupyter Notebooks have extension “.ipynb”



Interactive versus Script

- Interactive
 - - You type directly to Python one line at a time and it responds
- Script
 - - You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the statements in the file



Constants

- Fixed values such as numbers, letters, and strings, are called “constants” because their value does not change
- Numeric constants are as you expect
- String constants use single quotes (') or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

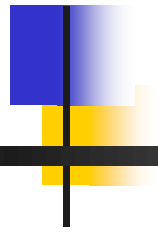


Reserved Words

- You cannot use reserved words as variable names / identifiers

False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
and	del	global	not	with
as	elif	try	or	yield
assert	else	import	pass	
break	except	in	raise	

<https://docs.python.org/3/library/functions.html#input>



Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

Good: spam eggs spam23 _speed

Bad: 23spam #sign var.12

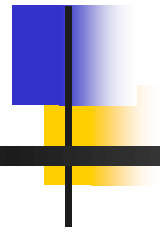
Different: spam Spam SPAM



Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

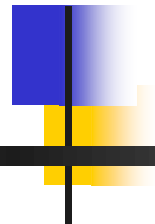


Operator Precedence Rules

- Highest precedence rule to lowest precedence rule:
 - Parentheses are always respected
 - Exponentiation (raise to a power)
 - Multiplication, Division, and Remainder
 - Addition and Subtraction
 - Left to right


Parenthesis
Power
Multiplication
Addition
Left to Right



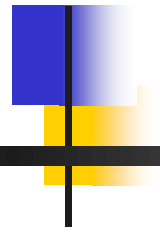


```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right



1 + 2 ** 3 / 4 * 5
↓
1 + 8 / 4 * 5
↓
1 + 2 * 5
↓
1 + 10
↓
11



Operator Precedence

- Remember the rules top to bottom
- When writing code - use parentheses
- When writing code - keep mathematical expressions simple enough that they are easy to understand
- Break long series of mathematical operations up to make them more clear



What Does “Type” Mean?

- In Python variables, literals, and constants have a “type”
- Python knows the difference between an integer number and a string
- For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```



Type Matters

- Python knows what “type” everything is
- Some operations are prohibited
- You cannot “add 1” to a string
- We can ask Python what type something is by using the `type()` function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```



Several Types of Numbers

- Numbers have two main types
 - Integers are whole numbers:
-14, -2, 0, 1, 100, 401233
 - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```



Type Conversions

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float
- You can control this with the built-in functions `int()` and `float()`

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```



Integer Division

- Integer division produces a floating point result

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```




String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object
to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
```

```
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'x'
```



User Input

- We can instruct Python to pause and read data from the user using the `input()` function
- The `input()` function returns a string

```
nam = input('Who are you? ')\nprint('Welcome', nam)
```

Who are you? Luis
Welcome Luis



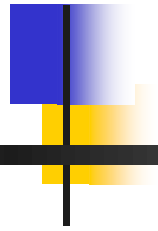
Converting User Input

- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function

```
inp = input('Europe floor?')  
usf = int(inp) + 1  
print('US floor', usf)
```


Europe floor? 0

US floor 1



Comments in Python

- Anything after a `#` is ignored by Python
- Why comment?
 - - Describe what is going to happen in a sequence of code
 - - Document who wrote the code or other ancillary information
 - - Turn on/off a line of code



```
# Get the name of the file and open it
name = input('Enter file:')
handle = open(name, 'r')

# Count word frequency
counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

# Find the most common word
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

# All done
print(bigword, bigcount)
```



Comparison Operators

- Boolean expressions ask a question and produce a Yes or No result which we use to control program flow
- Boolean expressions using comparison operators evaluate to True / False or Yes / No
- Comparison operators look at variables but do not change the variables

Python	Meaning
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	Greater than
!=	Not equal



Comparison Operators

```
x = 5
if x == 5 :
    print('Equals 5')
if x > 4 :
    print('Greater than 4')
if x >= 5 :
    print('Greater than or Equals 5')
if x < 6 : print('Less than 6')
if x <= 5 :
    print('Less than or Equals 5')
if x != 6 :
    print('Not equal 6')
```

Equals 5

Greater than 4

Greater than or Equals 5

Less than 6

Less than or Equals 5

Not equal 6

One-Way Decisions

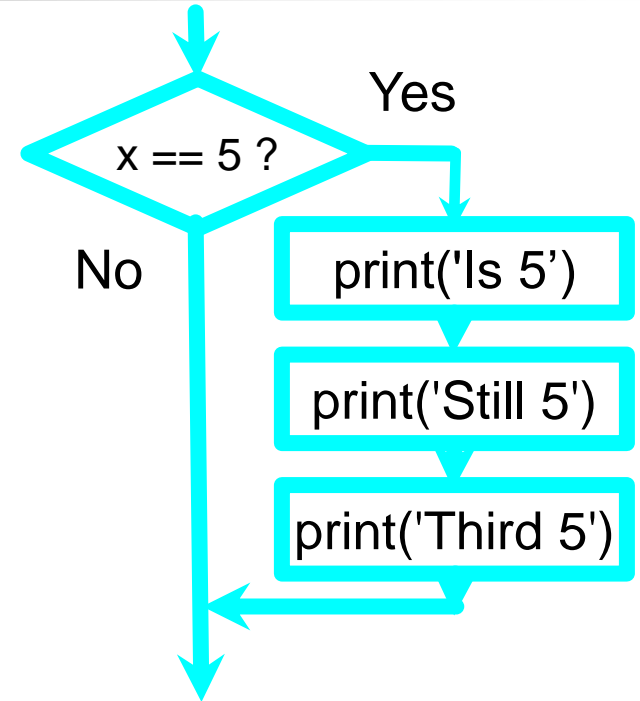
```
x = 5
print('Before 5')
if x == 5 :
    print('Is 5')
    print('Is Still 5')
    print('Third 5')
print('Afterwards 5')
print('Before 6')
if x == 6 :
    print('Is 6')
    print('Is Still 6')
    print('Third 6')
print('Afterwards 6')
```

Before 5

Is 5
Is Still 5
Third 5

Afterwards 5
Before 6

Afterwards 6

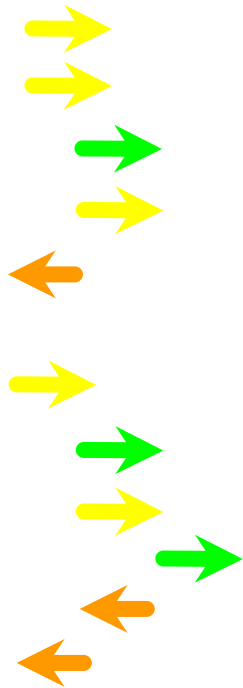




Indentation

- Increase indent after an if statement or for statement (after :)
- Maintain indent to indicate the scope of the block (which lines are affected by the if/for)
- Reduce indent back to the level of the if statement or for statement to indicate the end of the block
- Blank lines are ignored - they do not affect indentation
- Comments on a line by themselves are ignored with regard to indentation

increase / maintain after if or for
decrease to indicate end of block



```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')

for i in range(5) :
    print(i)
    if i > 2 :
        print('Bigger than 2')
    print('Done with i', i)
print('All Done')
```

***range(x)**

This function creates a 'list' of integers based on the value x

- x must be an integer
- The list will contain x integers 0 through x-1: [0,...,x-1]



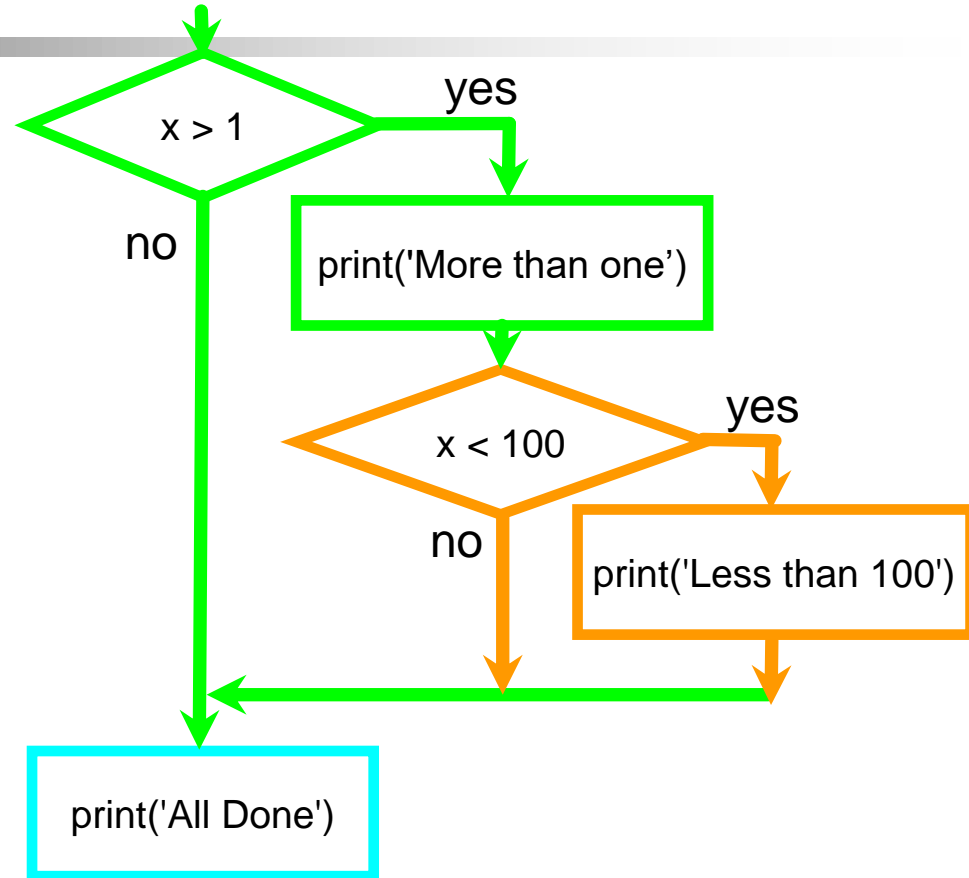
Think About begin/end Blocks

```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')
```

```
for i in range(5) :
    print(i)
    if i > 2 :
        print('Bigger than 2')
    print('Done with i', i)
print('All Done')
```

Nested Decisions

```
x = 42
if x > 1 :
    print('More than one')
    if x < 100 :
        print('Less than 100')
print('All done')
```

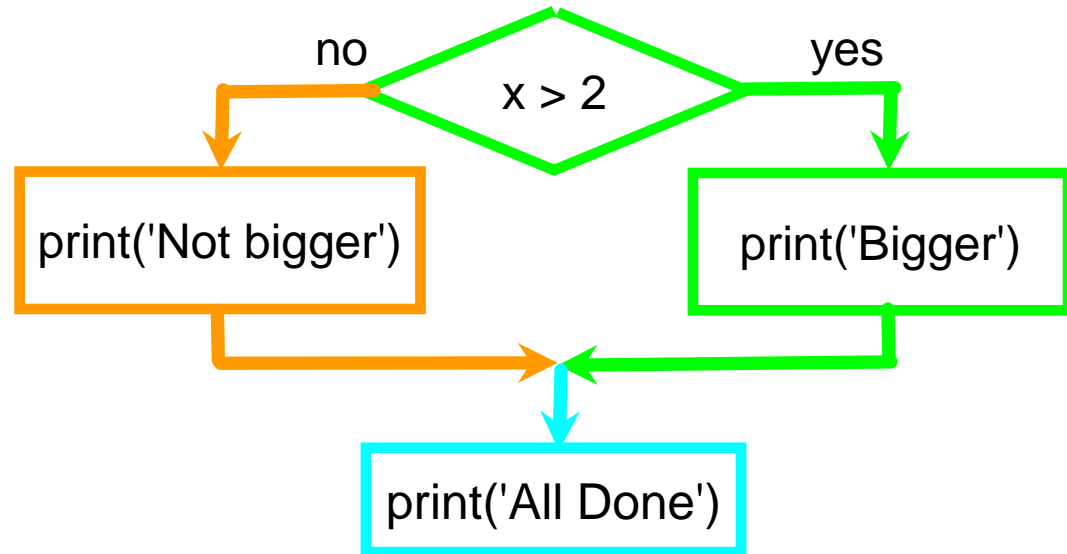


Two-way Decisions with else:

```
x = 4

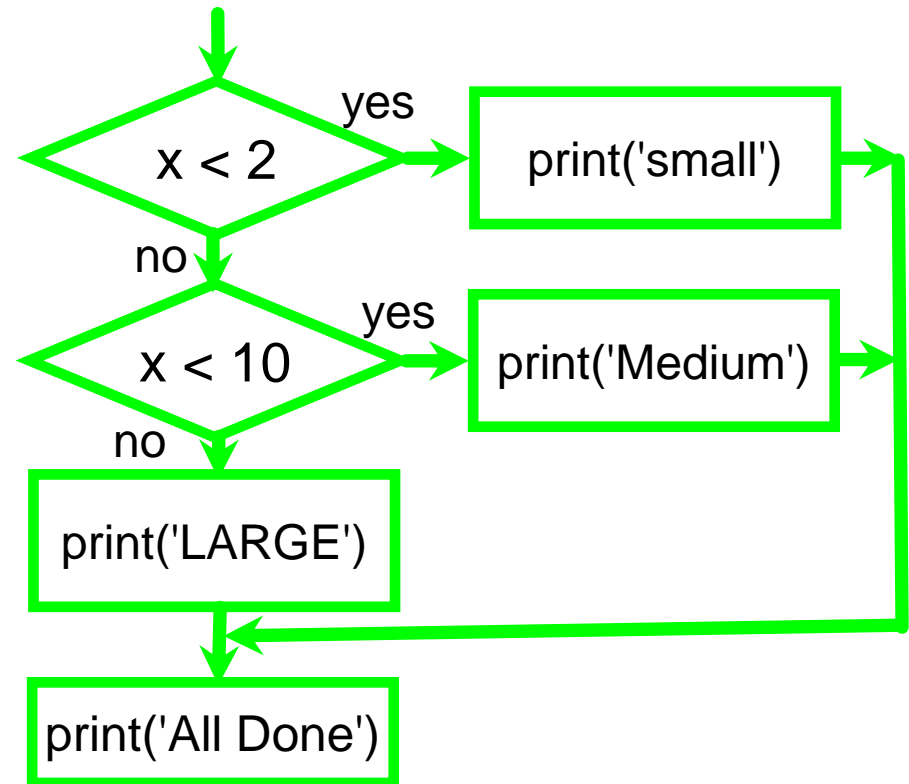
if x > 2 :
    print('Bigger')
else :
    print('Smaller')

print('All done')
```



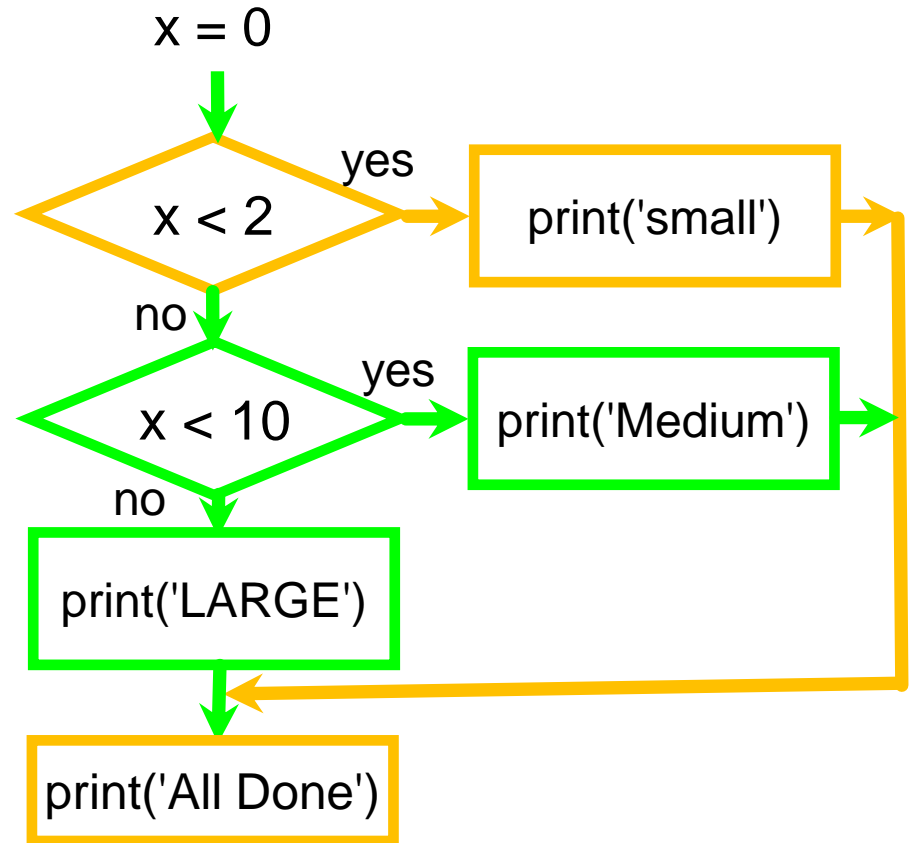
Multi-way

```
if x < 2 :  
    print('small')  
elif x < 10 :  
    print('Medium')  
else :  
    print('LARGE')  
print('All done')
```



Multi-way

```
x = 0
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```



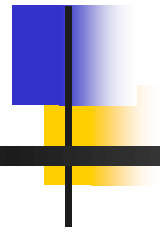


Multi-way

```
# No Else
x = 5
if x < 2 :
    print('Small')
elif x < 10 :
    print('Medium')

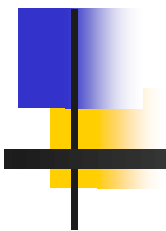
print('All done')
```

```
if x < 2 :
    print('Small')
elif x < 10 :
    print('Medium')
elif x < 20 :
    print('Big')
elif x < 40 :
    print('Large')
elif x < 100:
    print('Huge')
else :
    print('Ginormous')
```

The try / except Structure

- You surround a dangerous section of code with try and except (for validation or error trapping)
- If the code in the try works - the except is skipped
- If the code in the try fails - it jumps to the except section



```
$ python3 notry.py
```

```
Traceback (most recent call last):
```

```
File "notry.py", line 2, in <module>
```

```
istr = int(astr)ValueError: invalid literal  
for int() with base 10: 'Hello Bob'
```

```
$ cat notry.py  
astr = 'Hello Bob'  
istr = int(astr)  
print('First', istr)  
astr = '123'  
istr = int(astr)  
print('Second', istr)
```

Error, code
stops here



Let's use try / except!

```
astr = 'Hello Bob'
try:
    istr = int(astr)
except:
```



```
    istr = -1
```



Error Flag

```
print('First', istr)
```

```
astr = '123'
```

```
try:
    istr = int(astr)
except:
    istr = -1
```



```
print('Second', istr)
```

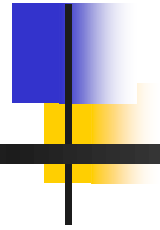


When the first conversion fails - it just drops into the except: clause and the program continues.

```
$ python tryexcept.py
First -1
Second 123
```

When the second conversion succeeds - it just skips the except: clause and the program continues.

Sample try / except – Entry Validation



```
rawstr = input('Enter a number:')
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print('Nice work')
else:
    print('Not a number')
```

```
$ python3 trynum.py
Enter a number:42
Nice work
```

```
$ python3 trynum.py
Enter a number:forty-two
Not a number
$
```